AFIT/GOR/ENS/98M-22

A GAME-THEORETIC IMPROVEMENT MODEL FOR STOCHASTIC NETWORKS: RELIABILITY VS. THROUGHPUT

THESIS

Jeffrey A. Schavland, Captain, USAF

AFIT/GOR/ENS/98M-22

DTIC QUALITY INSPECTED 4

Approved for public release; distribution unlimited

19980429 047

AFIT/GOR/ENS/98M-22

THESIS APPROVAL

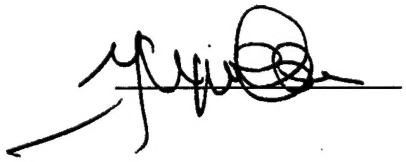
STUDENT: Captain Jeffrey A. Schavland CLASS: GOR-98M

THESIS TITLE: A Game-Theoretic Improvement Model for Stochastic Networks: Reliability vs. Throughput

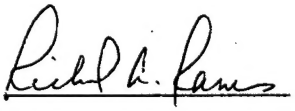
DEFENSE DATE: 19 February 1998

COMMITTEE: NAME/TITLE/DEPARTMENT SIGNATURE

Advisor Yupo Chan
Professor of Operations Research
Department of Operational Sciences
Air Force Institute of Technology



Reader Richard A. Raines, Major, USAF
Assistant Professor of Electrical Engineering
Department of Electrical and Computer Engineering
Air Force Institute of Technology



AFIT/GOR/ENS/98M-22

A GAME-THEORETIC IMPROVEMENT MODEL FOR STOCHASTIC NETWORKS: RELIABILITY VS. THROUGHPUT

THESIS

Presented to the Faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Operations Research

Jeffrey A. Schavland, B.S.

Captain, USAF

March 1998

Approved for public release; distribution unlimited

Acknowledgments

excusiatio propter infirmitatem
—the Medieval author's apology for the weakness of his talents

I still do not understand fully by what good fortune I happened to attend Professor Yupo Chan's elective on multiple criteria decision making in the spring of 1997—I wasn't even registered to take the class. The course introduced me to, in my humble opinion, the most interesting topic area in operations research and a wonderful teacher and thesis advisor. I thank Dr. Chan, for his time, patience, advice, and interest in this student's problems.

My gratitude is also extended to Major Richard Raines of the Department of Electrical and Computer Engineering and Dr. Lorin Lund and his associates in the R55 shop at the National Security Agency.

In the "It never hurts to ask" department, I must thank the people at Lindo Systems Inc., who, in my darkest hour of research, provided me—free of charge—a temporary copy of HyperLingo, an extraordinary software product.

Appreciation is expressed to Professor Douglas Shier of Clemson University and Dr. Miro Kraetzl of the Defence Science and Technology Organisation (Australian Ministry of Defence) for the software help.

My best wishes are extended to Eric Moyer of Wright State University, who, on very short notice, coded a C++ program to calculate two-terminal

reliability expressions using Shier's algebraic approach. (We both learned a valuable lesson in combinatorics!)

Despite periods of intense discomfort, I am grateful to the United States Air Force for a first class graduate education in operations research. I am convinced there is no better opportunity in the whole of the military than in-residence graduate education at the Air Force Institute of Technology.

Above all, I am eternally grateful for the love and understanding demonstrated by my wife, Adrienn, who married me only months before my assignment here.

Jeff Schavland

Table of Contents

Page

Acknowledgments ii

List of Figures vi

List of Tables viii

Abstract ix

I. Introduction 1
- Background 1
- Research Problem 2
- Scope 3
- Notation 5

II. Literature Review 7
- Network Types and Components 7
- Deterministic Flow 8
- Reliability 8
- Expected Flow 10
- Vulnerability 11
- (Reliability) Damage Utility 13
- Research Heritage 16

III. Models and Methodology 18
- On a Throughput-based Damage Utility 18
- A Quick Refresher on Game Theory 21
- Flow Damage Utility with a Hardening Game 23
- Mulicriteria Optimization: Reliability *and* Flow Damage Utility 25
- Summary of Flow Damage Utility Development and MCO Approach 26

IV. Example Analysis on Sample Network 29
- Description of Sample Network 29
- Application of Models to Sample Network 29
- Costing Capacity vs. Reliability Improvements 32
- Multiple Criteria Optimization—Flow *vs.* Reliability Damage Utilities 35
- Analysis of Efficient Frontiers 37

Value of Hardening 40
Value of Model without Network Improvement 42

V. Work on Larger Networks 44
Introduction 44
Computational Considerations 44
Networks A, B, and C 46
Model C Applied to Networks A, B, and C 50
Model D Applied to Network A 51

VI. Conclusions and Recommendations 54
Summary 54
Conclusions 54
Future Work 55

Appendix A: LINGO Input/Output Files for Sample Network 56

Appendix B: LINGO Input/Output Files for Networks A, B, and C 88

Appendix C: Manual and Computer Calculations of Reliability Expressions 143

Bibliography 176

Vita 180

List of Figures

Figure		Page
1.	Research Heritage	17
2.	Payoff Matrix for "Rock, Paper, Scissors"	22
3.	Payoff Matrix for Sample Network (Model C)	23
4.	Single Criteria Throughput Models	26
5.	Comparison of Game-Theoretic Models	27
6.	Construction of the Game-Theoretic MCO Model (Model D)	28
7.	Sample Network	29
8.	Results from Maximization of Expected Flow (Model A)	30
9.	Results from Maximization of Flow Damage Utility without Hardening (Model B)	31
10.	Results from Maximization of Flow Damage Utility with Hardening (Model C)	31
11.	Flow Damage Utility vs. Reliability Budget	34
12.	Flow Damage Utility vs. Capacity Budget	35
13.	Efficient Frontier: "Expensive" Capacity	35
14.	Efficient Frontier: "Neutrality" between Capacity and Reliability	36
15.	Efficient Frontier: "Cheap" Capacity	36
16.	Distances Between Efficient Design Points and Ideal	39
17.	Distances Between Efficient Design Points and Ideal	39
18.	Distances Between Efficient Design Points and Ideal	40
19.	Calculation of the Value of Hardening	41

20. V_{rel}-based Hardening Defensive/Offensive Strategy for Sample Network without Improvement Budget (Model 3) 42

21. V_{ef}-based Hardening Defensive/Offensive Strategy for Sample Network without Improvement Budget (Model C) 43

22. Network A 47

23. Network B 48

24. Network C 49

25. Network A: Efficient Frontier 52

26. Network A: Efficient Frontier Analysis 52

List of Tables

Table	Page
1. Network A Results (Model C)	50
2. Network B Results (Model C)	50
3. Network C Results (Model C)	51

Abstract

Prescriptive models used to allocate resources for network improvement traditionally have used reliability or flow as measures of effectiveness (MOEs). Such metrics do not give value to efforts that make a component more difficult to exploit. This study developed an entirely new MOE for stochastic network improvement, flow damage utility, which uses a two-person, zero-sum, non-cooperative game to optimize a probabilistic network for an estimate of expected flow minus performance degradation after a worst-case component loss. A multiple criteria optimization problem that uses flow damage utility and an analogous, previously developed metric for the reliability problem is used to capture the strategic competition between the network defender and attacker and shows promise of finding "value-free" improvement/defensive strategies in the context of Steuer's reverse filtering as applied to the generated efficient frontier. Irrespective of unit costs of reliability vs. bandwidth improvement, a "value-free" solution may be obtained, and the actual value of information security may be imputed from these game-theoretic models. Examples of analysis on four different networks are presented.

A GAME-THEORETIC IMPROVEMENT MODEL FOR STOCHASTIC NETWORKS: RELIABILITY VS. THROUGHPUT

I. Introduction

> In many industries, firms bear little resemblance to the passive bodies portrayed in the traditional economics textbook. Instead managers try to anticipate the actions of others—whether they be competitors, suppliers or customers—and influence those actions to their advantage.
>
> —*The Economist* on Game Theory, 24 January 1998 [14]

Background

In the world of stochastic networks, widely employed to model communication and distributed computer systems, it already has been demonstrated that maximization of aggregate statistical reliability, by itself, is a suboptimal improvement strategy in anticipation of an attack by an intelligent agent [21]. In the present Information Age, in which the importance of communication and computer networks grows unabated, methods to reduce the true vulnerability of networks while increasing their capabilities will be vital for individual security, economic health, and national defense [24].

The Economist's evaluation, captured in the quotation above, is *exactly* the situation faced today by not only those in the business community who

wish to move past traditional models of market behavior which assume *passive* competitors—but also by those within the Department of Defense who wish to defend assets against attacks by intelligent, dynamic adversaries in an era of declining budgets. As Lyle demonstrated, traditional network performance metrics such as reliability and survivability are *not* game-theoretic. They do not take into consideration the anticipated behavior of an intelligent adversary [21]. How can the actions the enemy be anticipated and thereby acted upon? This question is central to the work presented herein.

Research Problem

Specifically, this research will:

- develop a new, game-theoretic, flow-based network performance metric to complement the previously developed, reliability-based "damage utility" by Lyle,
- demonstrate the use of both game theoretic metrics in a multiple criteria game to capture the strategic competition for information security between network defender and attacker,
- apply both the single criteria (flow) and multiple criteria (flow and reliability) models to networks of realistic size, geometry, and complexity to be of interest to the Department of Defense and the civilian sector interested in information security.

Scope

Chapter II contains a review of the scientific literature relevant to the subject under investigation in addition to definitions of key terms and concepts in the area—examples of which are reliability, throughput, expected flow, and damage utility. Particular attention will be paid to recapitulating Lyle's work involving both a new metric, called (reliability) damage utility, and his concept of a hardening game played on a network.

Chapter III describes models and methodology for both a single, game-theoretic, flow-based metric and the combination of two game-theoretic metrics (flow- and reliability-based) in a multiple criteria game. The two-person, zero-sum, non-cooperative game *itself* is introduced first as a child's game and then adapted to capture the strategic competition between a network defender and attacker in the struggle for information security.

Chapter IV demonstrates the models and methods of Chapter III on a small, sample network. Furthermore, the efficient frontier generated by the multiple criteria model (i.e., the tradeoff space between the game-theoretic flow and reliability metrics) is analyzed using a technique called reverse filtering, which, in the case of the network examined in the chapter, reveals a "value-free" improvement/defensive strategy. That is, irrespective of tradeoff philosophy between capacity (bandwidth) and reliability—whether totally compensatory or totally noncompensatory—there is a well-defined, dominant, equilibrium solution. An approach to avoid the problem of cost uncertainty

between component capacity vs. reliability improvement is demonstrated (promising "cost-free" analysis) as is a method to impute the value of hardening or making a component more difficult to exploit.

Chapter V looks at applications on larger networks more relevant to the Department of Defense. The single-criteria flow model is applied to three large networks, and multiple criteria optimization to one. The existence of a "value-free" solution, as seen in Chapter IV, appears here, as well.

Chapter VI provides a summary and conclusions, in addition to recommendations for further research.

Notation

The following notation is used throughout this thesis. It is presented here for easy reference.

B_{cap}	budget for component capacity improvement
B_{rel}	budget for component reliability improvement
B_{tot}	total budget for component improvement
c_{ij}	initial capacity of the directed arc from node i to node j
d_{ij}	improvement to c_{ij}
p_{ij}	initial probability that arc i-j is operational
p_k	initial probability that node k is operational
r_{ij}	improvement to p_{ij}
r_k	improvement to p_k
R_{ij}	two-terminal reliability after the removal of arc i-j
R_{st}	two-terminal reliability
s	source node
s_1	a constant (minimum value of R_{st} in a reduced feasible region multiple criteria nonlinear program)
s_2	a constant (minimum value of V_{rel} in a reduced feasible region multiple criteria nonlinear program)
t	sink or terminus node

V maximum (expected) flow or throughput

V_{ef} expected flow damage utility

V_{rel} reliability damage utility

x_{ij} flow through directed arc i-j

y_{ij} defensive (Blue) game variable ("hardness" given to arc i-j)

z_{ij} offensive (Red) game variable (shadow price of game constraint for arc i-j, interpreted as Red attack strategy)

II. Literature Review

> Human history is saturated with instances demonstrating the ceaseless effort to reduce and eliminate real or perceived vulnerability of every kind. Bonfires, body armor, fortresses, moats, [the] 'Great Wall of China,' medicines, and magic potions were developed to reduce or prevent harm. In a way, history is a record of Man's struggle to be free of the shackles of all forms of vulnerability.
>
> —Derek de Sa, "Vulnerability of Flow Networks," 1989 [13]

An enormous body of literature exists on the subject of networks and likewise on game theory. It does not follow, however, that the set containing the intersection of the two is populated—at least in 1998. Based on anecdotal information and a reading of the current literature, there does appear to be a renewed interest in game-theoretic applications to both business and military problems (ref. Chapter I), so the dearth of literature on the subject may only be temporary. In this chapter, necessary network topics are explained and Lyle's work on the reliability-based network game is recapitulated as compactly as possible. A treatment of the two-person, zero-sum, non-cooperative game model is handled in Chapter III.

Network Types and Components

A probabilistic graph $G=(V,E)$ may be used to represent a computer or communications network, where V is a set of nodes (vertices being the graph theoretic term) and E a set of *undirected* edges and/or *directed* arcs, denoting bi-directional and unidirectional communication links, respectively [11]. This

work will concern itself exclusively with graphs containing directed arcs only, called directed graphs or digraphs. For completeness, a mixed graph contains both arcs and edges, and the general term, graph, is used to describe graphs which contain only edges. Networks under consideration are stochastic, rather than deterministic, because of the failure probabilities associated with arcs.

Deterministic Flow

Networks may and commonly do have a lower and/or upper bound associated with each arc, constraining the amount of flow which may be transported through them. In communication networks, for example, these capacities correspond to link bandwidths. An important problem in operations research has been the maximum flow or "max" flow problem which is used to calculate the throughput of a network from a single source node, s, to a single sink or target node, t. Numerous algorithms have been developed to solve the max flow problem, notably Ford and Fulkerson's labeling method (which is a specific implementation of the out-of-kilter algorithm) and specialized primal simplex methods [5].

Reliability

Reliability is a measure of a network's ability to perform a desired operation. The common and traditional operation a network is expected to

perform is maintaining at least one open path from a single source to a single sink. Specific terms to describe this arrangement include two-terminal reliability or *s,t*-connectedness. Numerous other types of reliability exist, an example of which is all-terminal reliability, requiring at least one open path from every node to every other node in the network [11].

Reasonable and useful assumptions from the standpoint of model tractability are that each arc has a statistically independent, binomial probability associated with it and further that nodes do not fail. The latter assumption need not be a cause for concern. Any probabilistic node can be represented by a probabilistic arc connected to two failure-free nodes [27] [30].

Network reduction is one method used to calculate statistical reliability. Serial and parallel relationships are applied recursively until the network is reduced—through series and parallel paths—to a single component with a specified reliability function. This method's primary drawback, however, often is referred to as the "state-space explosion"; i.e., as the number of components (n) grows, the number of possible network states (k) grows exponentially ($k=2^n$)—making network reduction an *np*-hard problem [11].

Reliability is not to be confused with vulnerability, availability, survivability, safety, resilience, or stability. To be sure, the terms all relate to one another, but only through their careful definition do they become

useful to the network designer. It is important to note here perhaps the biggest drawback to reliability (and a clue as to why it plainly is not vulnerability): reliability specification requires *complete* knowledge of the failure characteristics of every system component, the impact of every possible combination of component failure on the system, and the exogenous environment's interaction with the system [13]. de Sa states: "Reliability analysis is limited in that it can only focus on those failures which are considered possible within the limits of the analyst's creative imagination and the given conditions. Unfortunately for the reliability analyst, knowledge of complex systems and real world processes is never complete [13]." A consequence of this limitation is that reliability analysis *cannot* be used to determine how well a network will fare against an intelligent adversary.

Expected Flow

In a stochastic network, the solution to the max flow problem is not of much interest. Here, *expected* flow is the analogous metric. As with the calculation of network statistical reliability, exact computation of expected flow runs into the "state-space explosion" problem for the same reasons. Given n probabilistic components, there are 2^n possible network states, *each* of which require a deterministic max flow calculation. An exact formula for expected flow is:

$$\text{expected throughput (exact)} = \sum_{k=1}^{2^n} F_k P_k$$

where F_k is max flow of network in state k

and

P_k is probability of being in state k [10]

The good news is that approximating bounds have been found which appear to be useful to the network designer. Yim, for example, has demonstrated a fairly tight lower bound on stochastic network throughput on a variety of large and small realistic networks [30]. The model, which uses a network's paths, rather than arcs, not only runs in a single linear program (LP), but also can be used prescriptively for network improvement.

Jansen *et al.* have generated efficient frontiers of reliability versus expected throughput on a variety of realistic networks using a multicriteria optimization approach [9] [17]. With these efficient frontiers, analysis of the tradeoff space may be performed using a compromise programming approach [Chan].

Vulnerability

The American Heritage College Dictionary defines "vulnerable" as "susceptible to physical injury" or "susceptible to attack [3]." de Sa draws five conclusions from the general definition of vulnerability:

- The definition of vulnerability is inextricably linked [to] and must follow from the definition of system "wounding" or harm, damage, and failure.

- Vulnerability has multiple attributes according to the different kinds of harm, exposure, and lack of protection relevant to the system.

- Vulnerability implies that the potential for harm, injury or damage originates from at least one exogenous source.

- Vulnerability analysis includes the evaluation of both static and dynamic characteristics of the system to defend itself, i.e., to prevent and eliminate (the consequences of) harm. A system unable to reduce the "wound" or recover from harm is correspondingly vulnerable. These are also the attributes of system resilience and survivability [13].

Sengoku *et al.* have proposed two related indices measuring the degree of influence of an edge on the vulnerability of a network [25]. They are:

$$\lambda(e_k) = \sum_{ij} g(v_i, v_j; e_k)$$

and

$$\lambda_n(e_k) = \frac{\sum_{ij} g(v_i, v_j; e_k)}{f(v_i, v_j)}$$

where

$v_i, v_j \in V$

$v_i \neq v_j$

$f(v_i, v_j) \neq 0$

Here, f is the value of a maximum flow from a vertex v_i to v_j, and g indicates the minimum value of the flow through arc e_k in the maximum flows. The latter is simply a normalized version of the former. From these measures of arc contribution to network vulnerability, this work later will investigate a general network improvement strategy which aims to minimize the "importance" of every arc while maximizing expected throughput.

Bagga *et al.*, Malashenko *et al.*, and de Sa have all written recent summaries of the literature on the vulnerability of networks [4] [22] [13]. The vast majority of the scientific papers on the subject of vulnerability concern deterministic networks, and many of the few which address the vulnerability of *stochastic* networks use reliability and vulnerability interchangeably.

(Reliability) Damage Utility

This next section is a more thorough and detailed review of Lyle's work in the area invulnerability, a multicriteria measure of two-terminal reliability vs. a new metric called "damage utility [13]." Lyle's damage utility, hereafter referred to as reliability damage utility to avoid confusion with an upcoming flow-based metric, is a function of the statistical reliability of a network given that a component has been destroyed minus the statistical reliability of the network before the component was destroyed.

If network statistical reliability were the only goal, a single criteria optimization problem could be formulated (Model 1) to maximize R_{st}. Model 1 answers the question: "How should resources for improvement of component reliability be allocated in order to maximize the probability that there is at least one operational path between source and sink?" This is the classical, non-game-theoretic approach to reliability-based network improvement.

Model 1

$\max Rst(\mathbf{p}+\mathbf{r})$

subject to

$p_{ij} + r_{ij} \leq 1 \qquad \forall \text{ arcs } ij$

$\sum_{\text{arcs}ij} p_{ij} r_{ij} \leq B_{rel}$

all variables ≥ 0

all of which define the space $\mathbf{X}_1$, $\mathbf{r} \in \mathbf{X}_1$

Model 2 is an intermediate step on the way to a completely game-theoretic model. Given a "Blue" network, a rational and informed "Red" adversary would attack and destroy the network component which causes the greatest damage, defined as the statistical reliability of the network given that a component has been destroyed minus the statistical reliability of the network before the component was destroyed. Let $V_{rel} = min\{1 - R_{st} + R_{ij}\}$, where V_{rel} is damage utility (ranging from 0 to 1) and R_{ij} is the network statistical reliability after arc ij is rendered inoperative.

Model 2

$\max V_{rel}$

subject to

$V_{rel} \leq R_{ij} - R_{st} + 1 \qquad \forall \text{ arcs } ij$

$\mathbf{r} \in \mathbf{X}_1$

Model 3 completes the game-theoretic model by including the option for Blue to "harden" network components in addition to improving their reliabilities. The y_{ij} term represents the proportion of hardening effort into a single component with the sum of all ys equal to 1. Placing all of the hardening effort onto a single component renders it completely invulnerable

to an attack. No accounting for the cost of hardening is made by this model explicitly—its value, however, may be imputed by comparing the budgets required to reach a target reliability damage utility with both Model 2 and 3, the difference (3's budget minus 2's) being the value of hardening. The concepts of the hardening game and the imputed value of hardening are given a more complete treatment in Chapters III and IV, respectively.

Model 3

$\max V_{rel}$

subject to

$V_{rel} \le (R_{ij} - R_{st} + 1)(1 - y_{ij}) + y_{ij} \qquad \forall \text{ arcs } ij$

$\sum_{\text{arcs } ij} y_{ij} = 1$

$\mathbf{r} \in \mathbf{X}_1$

Model 4 is a multicriteria optimization (MCO) model to maximize V_{rel} and R_{st} simultaneously. To solve this MCO program in practice (generate the efficient frontier), the minimum statistical reliability criterion function is varied parametrically from its minimal value (such that V_{rel} is at its maximum) to 1, and B_{rel} is held constant (where $R_{st} = 1$ in Model 1). Let this constraint be $R_{st} \ge s_1$, a constant. This approach to solving a MCO problem is known as the constraint-reduced feasible region method.

Model 4

$\max V_{rel}$

$\max R_{st}$

subject to

$V_{rel} \le (R_{ij} - R_{st} + 1)(1 - y_{ij}) + y_{ij} \qquad \forall \text{ arcs } ij$

$\sum_{\text{arcs } ij} y_{ij} = 1$

$\mathbf{r} \in \mathbf{X}_1$

Lyle concluded that statistical reliability and reliability damage utility are very different measures of effectiveness which produce different prescriptions for optimal improvement, the value of hardening components can be imputed from the results of the models, and minimal decision-maker input is needed (that is, Model 4 produces a "value-free" or nearly value-free solution in the context of a procedure to analyze efficient frontiers called reverse filtering which will be discussed in detail in Chapter IV).

Research Heritage

Figure 1 is an attempt pictorially to summarize the contributions of prior work and to present a taxonomy of the research area.

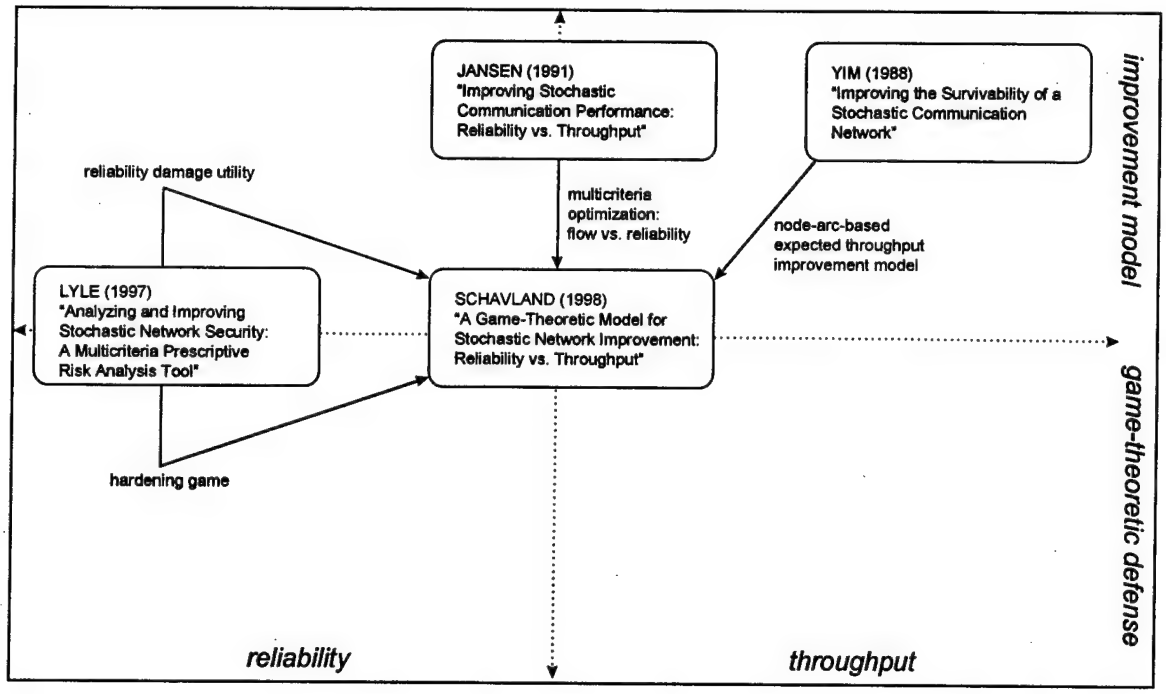


Figure 1. Research Heritage.

Figure 1 divides network research in the area of network improvement models into reliability and throughput (or flow) areas. It also includes an improvement model/game-theoretic defense axis. Since Lyle's work, for example, was a network improvement model which incorporated a network security game, his "box" straddles the axis.

III. Models and Methodology

> If there ever was an area of management science/operations research this exciting and challenging but fraught with deceptions and pitfalls, it is multiple criteria optimization. The field is fascinating because it clearly is an art as well as a science. When studying multiple criteria optimization, we learn how not to be naïve. We learn about where the "bodies are buried" in mathematical programming and how to be innovative in overcoming the variety of solution set and solution procedure difficulties that may arise. To me, multiple criteria optimization is a breath of fresh air: We can now openly admit that a problem has multiple objectives when it possesses multiple conflicting criteria. There is no need to ignore or gloss over the fact. We can deal with multiple objectives head on, because we now have the tools to solve large-scale multiple criteria optimization problems.
>
> —Ralph E. Steuer, *Multiple Criteria Optimization*, 1986 [28]

On a Throughput-Based Damage Utility

At the close of Chapter II, work was presented on a reliability-based damage utility for and its associated hardening game on a probabilistic network. However, as interesting as reliability and reliability-based metrics are, they only can indicate the likelihood of finding, in the case of two-terminal reliability, an open path between source and sink. What about flow? Perhaps counter-intuitively, past research has demonstrated that expected throughput and reliability, though clearly related, are not co-linear—i.e., given limited resources for the reliability improvement of network components, a trade-off does exist between the two performance metrics (Chapter II). Flow and reliability form the two important families of network performance metrics.

To reiterate, in a probabilistic network, the measure of throughput is *expected* (maximum) flow, since arcs are subject to random failures. An *exact* calculation of expected flow for the small sample network used in Chapter IV (Figure 7) would require evaluation of $2^7 = 128$ different states! For the purposes of this research, a proxy for expected flow is used; namely, maximum flow through the network, given that the capacity of each arc assumes its expected value. This measure forms an absolute upper bound. An NLP to maximize this upper bound on expected flow appears in Model A, in which separate budgets for reliability and capacity improvement are used. (The subject of a single budget for both kinds of network improvement and the problems it causes is covered later in this chapter.)

Model A

$\max V(\mathbf{d}, \mathbf{r})$

subject to

$\sum_{j=s} x_{jk} - V = 0$		(flow out of source)
$\sum_{k \neq t} x_{jk} - \sum_{i \neq s} x_{ij} = 0$		(flow balance at nodes)
$V - \sum_{j=t} x_{ij} = 0$		(flow into sink)
$x_{ij} \leq (c_{ij} + d_{ij})(p_{ij} + r_{ij})$	$\forall$ arcs ij	(expected value of individual arc capacity)
$\sum_{\text{arcs}ij} c_{ij} d_{ij} \leq B_{cap}$		(capacity budget)
$\sum_{\text{arcs}ij} p_{ij} r_{ij} \leq B_{rel}$		(reliability budget)
$p_{ij} + r_{ij} \leq 1 \forall$ arcs ij		(reliability limit)

all variables ≥ 0

Let this set of constraints form $\mathbf{X}_2$; $\mathbf{d}, \mathbf{r} \in \mathbf{X}_2$.

A common assumption is used in the generation of the budget constraints; namely, the cost of improving the reliability/capacity of a component is linear and equal to its current reliability/capacity. In the case of reliability, for example, it makes sense that it costs much more to improve a component's operational reliability from *90* to 95% than from, say, *60* to *65*%. Is this assumption perfect? Of course not. When the concepts presented in this thesis are translated into "nuts and bolts" applications (i.e., *domain-specific* problems), by all means, better cost constraints, if available, should be used. In their absence, however, the approach presented herein is conventional and, *at least*, offers a starting point for analysis.

A damage utility analogous to that developed for reliability is derived from Model A easily—call this Model B, in which the network is improved to maximize an estimate of its upper bound on expected flow after a "worst-case" link is destroyed. As with the case of V_{rel} and R_{st}, it is easy to demonstrate that V and V_{ef} are two different MOEs; in practice (Chapters IV and V), each model commits resources quite differently.

Model B	
max V_{ef}	
subject to	
$V_{ef} \leq V - x_{ij}$	$\forall$ arcs ij
$\mathbf{d}, \mathbf{r} \in \mathbf{X}_2$	

Model B is conservative, because it is guaranteed to overestimate the amount of damage (loss of system throughput) to the upper bound on expected flow, as the following inequality reveals:

$$0 \leq g(v_s, v_t; e_{ij}) \leq x_{ij} \leq (p_{ij} + r_{ij})(c_{ij} + d_{ij}) \leq (c_{ij} + d_{ij})$$

Sengoku's concept of a vulnerability metric is borrowed here and modified somewhat. g in this context is the minimum flow from node i to j (i.e., through arc i-j) such that the upper bound on expected flow from s to t is at its maximum value.

A Quick Refresher on Game Theory

First and foremost, this research examines the utility of a *game-theoretic* approach to network improvement. To be precise, the "game" in "game-theoretic" refers to a two-person, zero-sum, non-cooperative game. Why?

In the context of a Blue network defender *versus* a Red network attacker, only two sides of this conflict are possible, hence a *two-person* game. Irrespective of the criterion of network performance under consideration, Blue's loss is Red's gain and *vice versa*, hence a *zero-sum* game. For example, if Red's attack on Blue's network results in two-terminal reliability's dropping to, say, 0.75 from 0.90, blue's loss of 0.15 reliability "units" is exactly red's gain. Both sides are enemies of one another, or, in the more

diplomatic language of game theory, their "interests are opposed," hence a *non-cooperative* game [6] [23].

A simple example of a two-person, zero-sum, non-cooperative game is the childhood favorite, "Rock, Paper, Scissors." Each player simultaneously shows the hand symbol of either a rock, piece of paper, or pair of scissors, where a rock breaks a pair of scissors, a pair of scissors cuts the piece of paper, and the piece of paper covers the rock. A payoff matrix corresponding to the game is as follows:

		Red Player		
Blue Player	*Hand Signal*	Rock	Paper	Scissors
	Rock	0	-1	+1
	Paper	+1	0	-1
	Scissors	-1	+1	0

Figure 2. Payoff Matrix for "Rock, Paper, Scissors."

A linear program can be derived from the payoff matrix to calculate the best game strategy [29]. The expected value of each column of the payoff matrix corresponds to a constraint in the following LP:

$$\max U$$

subject to

$$U \leq x_{paper} - x_{scissors}$$

$$U \leq x_{scissors} - x_{paper}$$

$$U \leq x_{rock} - x_{paper}$$

$$x_{rock} + x_{paper} + x_{scissors} = 1$$

$$x_{rock}, x_{paper}, x_{scissors} \geq 0$$

The solution to the above LP happens to be

$$x_{rock} = x_{paper} = x_{scissors} = \frac{1}{3}$$

which means that the best strategy for a player to adopt is to randomly select one of the three forms for every play. In the language of game theory, this kind of solution is referred to as a *mixed*, as opposed to a *dominant* (where one of the game variables is equal to 1 and all other equal to 0), strategy.

Flow Damage Utility with a Hardening Game

The first step in playing the hardening game with flow is to generate a payoff matrix. Figure 2 is such a matrix for the sample network (Figure 3 in the next chapter).

arc	***s-1***	***s-2***	***1-3***	***1-4***	***2-4***	***3-t***	***4-t***
s-1	V	$V - x_{s-2}$	$V - x_{1-3}$	$V - x_{1-4}$	$V - x_{2-4}$	$V - x_{3-t}$	$V - x_{4-t}$
s-2	$V - x_{s-1}$	V	$V - x_{1-3}$	$V - x_{1-4}$	$V - x_{2-4}$	$V - x_{3-t}$	$V - x_{4-t}$
1-3	$V - x_{s-1}$	$V - x_{s-2}$	V	$V - x_{1-4}$	$V - x_{2-4}$	$V - x_{3-t}$	$V - x_{4-t}$
1-4	$V - x_{s-1}$	$V - x_{s-2}$	$V - x_{1-3}$	V	$V - x_{2-4}$	$V - x_{3-t}$	$V - x_{4-t}$
2-4	$V - x_{s-1}$	$V - x_{s-2}$	$V - x_{1-3}$	$V - x_{1-4}$	V	$V - x_{3-t}$	$V - x_{4-t}$
3-t	$V - x_{s-1}$	$V - x_{s-2}$	$V - x_{1-3}$	$V - x_{1-4}$	$V - x_{2-4}$	V	$V - x_{4-t}$
4-t	$V - x_{s-1}$	$V - x_{s-2}$	$V - x_{1-3}$	$V - x_{1-4}$	$V - x_{2-4}$	$V - x_{3-t}$	V

Figure 3. Payoff Matrix for Sample Network (Model C).

The payoff table has this interpretation: each row corresponds to Blue's making a single component invulnerable to removal (by any means); each column corresponds to Red's removing a component from the network with the caveat that if Blue has hardened a component, Red may attack it, but the removal attempt will be unsuccessful.

From this matrix, constraints for the value function may be generated by calculating the expected value of each column. For example, the expected value of column 1 is:

$$
\begin{aligned}
& Vy_{s-1} + (V - x_{s-1})(y_{s-2} + y_{1-3} + y_{1-4} + y_{2-4} + y_{3-t} + y_{4-t}) \\
&= Vy_{s-1} + (V - x_{s-1})(1 - y_{s-1}) \\
&= Vy_{s-1} + V - Vy_{s-1} - x_{s-1} + x_{s-1}y_{s-1} \\
&= V - x_{s-1}(1 - y_{s-1})
\end{aligned}
$$

Model C, whose game constraints are derived from the above payoff matrix, is as follows:

Model C

max V_{ef}

subject to

$V_{ef} \leq V\text{-}x_{ij}(1\text{-}y_{ij})$ $\quad \forall$ arcs ij (game constraints)

$\sum_{\text{arcs}\,ij} y_{ij} = 1$

$\mathbf{d}, \mathbf{r} \in \mathbf{X}_2$

An interesting result computationally from Model C is that the shadow prices (z_{ij} variables) corresponding to each of the game constraints sum to 1. An interpretation of these shadow prices is that they represent an optimal offensive (Red) attack strategy, which will be discussed further in Chapter IV. The defensive game variables, y_{ij} terms, sum to 1 through game construction (a constraint in the NLP).

Multicriteria Optimization: Reliability *and* Flow Damage Utility

Taking reinforcement from the Steuer quote at the beginning of this chapter, it is no longer necessary to pretend that a problem does not possess multiple, perhaps conflicting, criteria, when it, in fact, does. A "perfect" network would be optimized for both the game-based reliability *and* flow damage utilities. Model D is the MCO model. Observe that the two separate problems are linked by the reliability budget constraint and reliability improvement variables.

Model D

$\max V_{ef}$

$\max V_{rel}$

subject to

$V_{ef} \le V\text{-}x_{ij}(1\text{-}y_{ij})$ $\quad \forall$ arcsij (game constraints)

$V_{rel} \le (R_{ij} - R_{st} + 1)(1 - y_{ij}) + y_{ij}$ $\quad \forall$ arcsij (game constraints)

$\sum_{\text{arcs}ij} y_{ij} = 1$

$\mathbf{d}, \mathbf{r} \in \mathbf{X}$

$\mathbf{X} \equiv \mathbf{X}_1 \cup \mathbf{X}_2$

Model D can be solved using a constraint reduced feasible region approach to the multiple objective nonlinear program. Replace the criterion function, max V_{rel}, with the constraint, $V_{rel} \ge s_2$. Unlike Models A-C, Model D is not used to find *an* answer. By parametrically varying the constant, s_2, from its largest value such that V_{ef} is at its maximum without the V_{rel} constraint (slack associated with the constraint is not equal to zero)—all the

way to V_{rel}'s maximum value, a frontier of non-inferior (efficient, non-dominated, Pareto-optimal) solutions may be generated.

Summary of Flow Damage Utility Development and Multicriteria Optimization Approach

This section is an attempt to put into a compact and summary form the models discussed thus far. Figure 4 presents a summary of the development of flow damage utility.

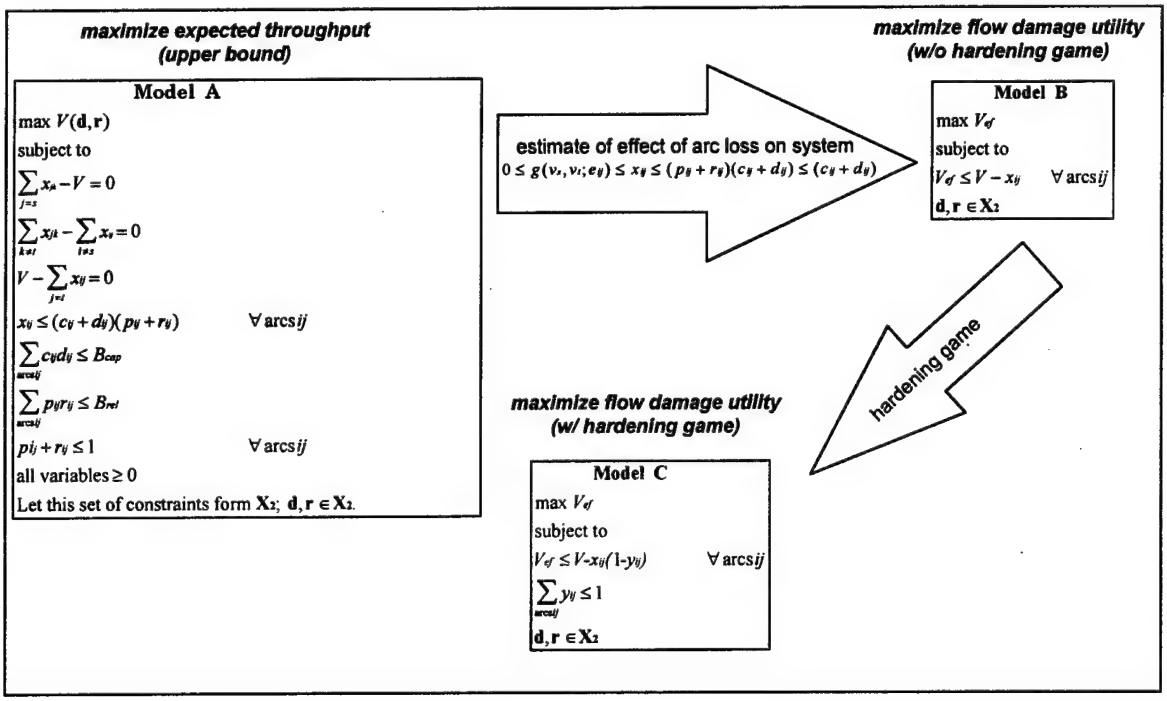


Figure 4. Single Criteria Throughput Models.

Figure 5 is a side-by-side comparison of the child's game used for an introduction to the two-person game and the network hardening game used in the context of flow damage utility.

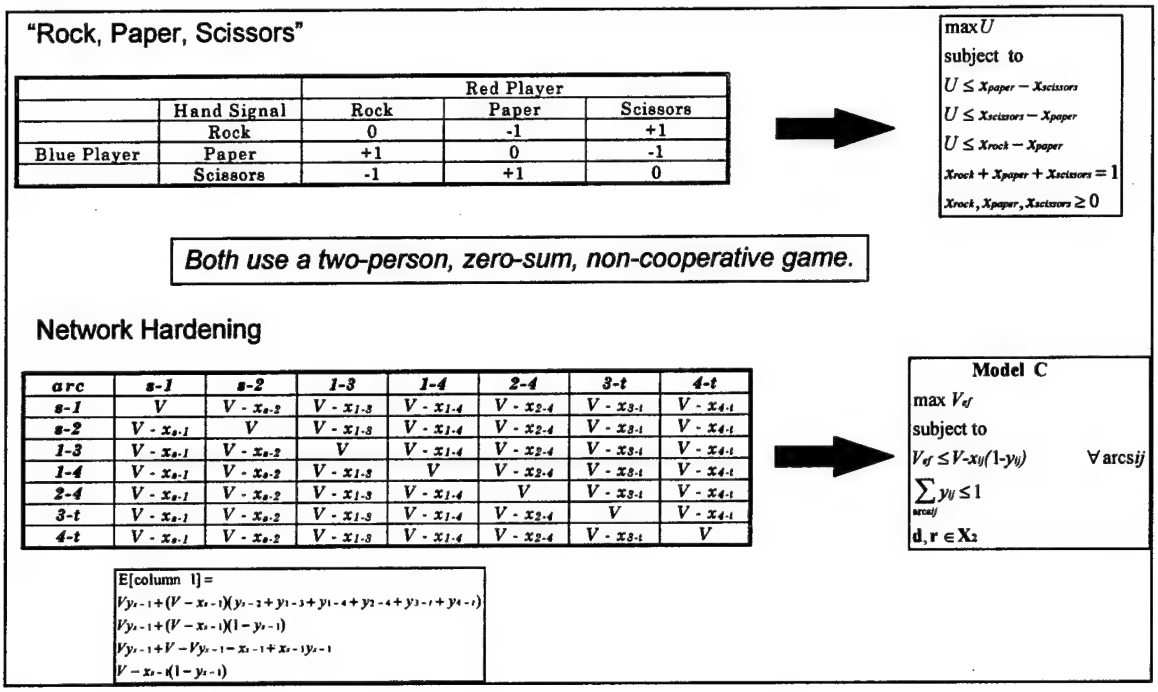


Figure 5. Comparison of Game-Theoretic Models.

Figure 6 illustrates the construction of the game-theoretic, multicriteria optimization model (Model D). Reliability damage utility and flow damage utility are the two criterion functions to be maximized subject to the intersection of the feasible regions from Models 3 and C. Finally, as the figure indicates, the constraint reduced feasible region is used to actually solve this multicriteria NLP.

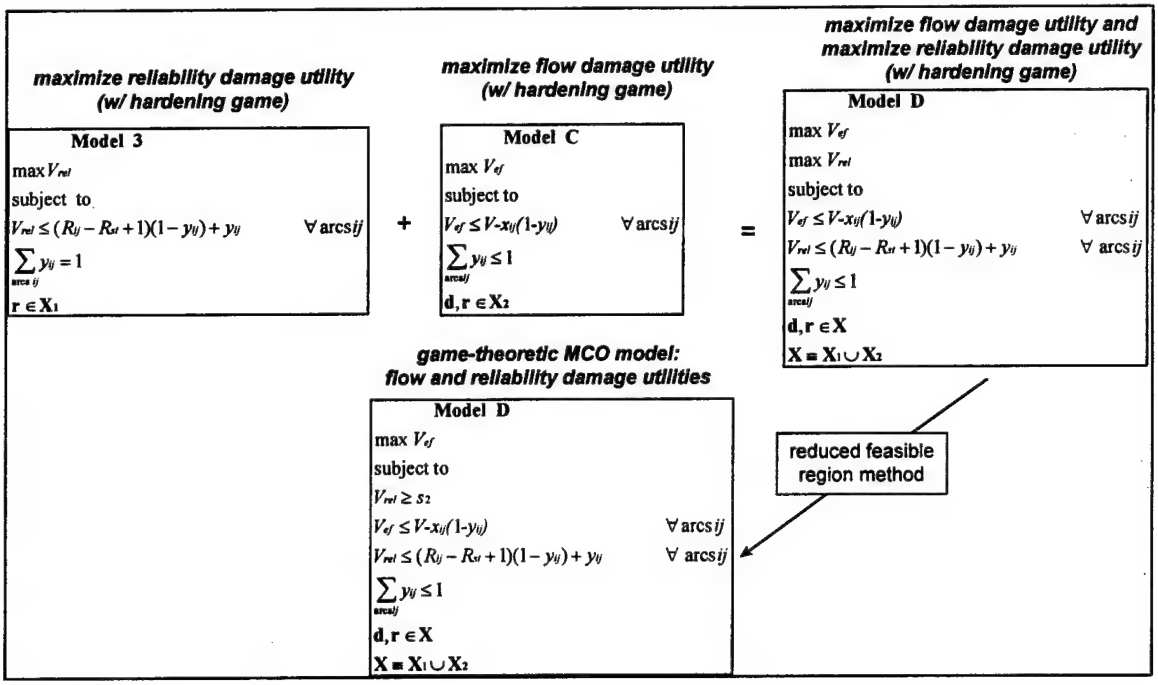


Figure 6. Construction of the Game-Theoretic, Multicriteria Optimization Model (Model D).

IV. Example Analysis on Sample Network

Description of Sample Network

The probabilistic network shown in Figure 7 will be used to illustrate the models and methodology under discussion. The network has a single source, *s*, and a single sink, *t*. The number adjacent to each arc indicates its maximum capacity. In the case of this network, arcs do not have minimum capacities. Finally, every arc has an operational probability of *0.8*. Nodes here are not subject to failure. The reader is referred to the discussion in Chapter II regarding translation between node and arc failure on probabilistic networks.

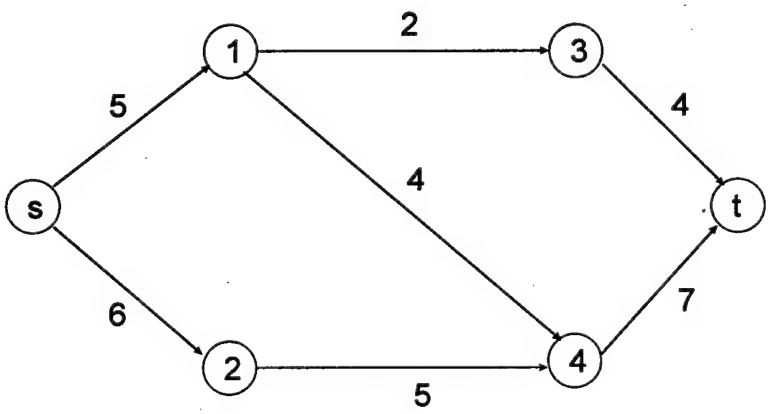


Figure 7. Sample Network.

Application of Models to Sample Network

Throughout this thesis, LINGO, the popular, powerful, and easy-to-use mathematical modeling software by LINDO Systems Inc. is employed to solve

all mathematical programs—both non-linear and linear [20]. All models are presented in the Appendices in LINGO's self-explanatory format and will run "as-is"—no editing should be necessary. The reader is invited to download a free evaluation copy of LINGO at http://www.lindo.com. Detailed input/output files for these sample runs are included in Appendix A.

Figures 8-10 display the improvement strategies from Models A-C, respectively. Italicized numbers are initial capacity plus capacity improvement. Non-italicized numbers are initial reliability plus reliability improvement. Boxed percentages are game variable (hardening) values.

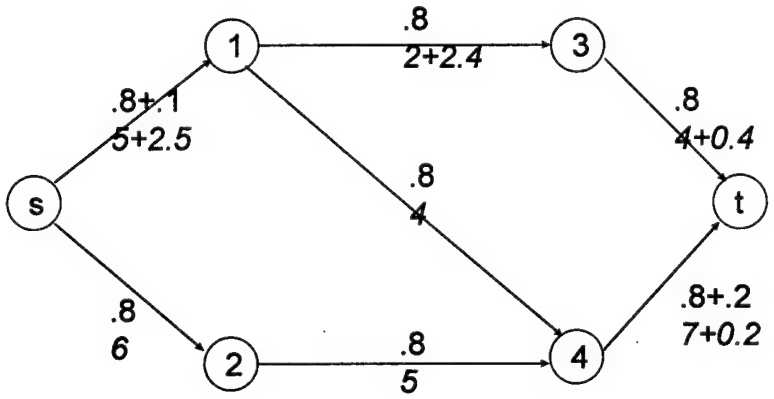


Figure 8. Results from Maximization of Expected Flow (Model A).

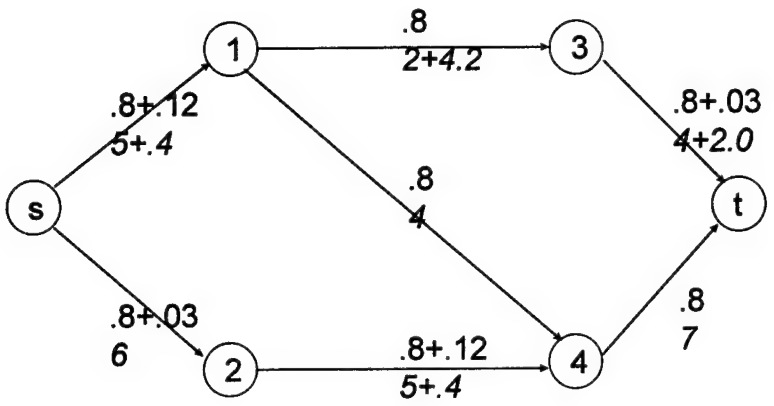


Figure 9. Results from Maximization of Flow Damage Utility without Hardening (Model B).

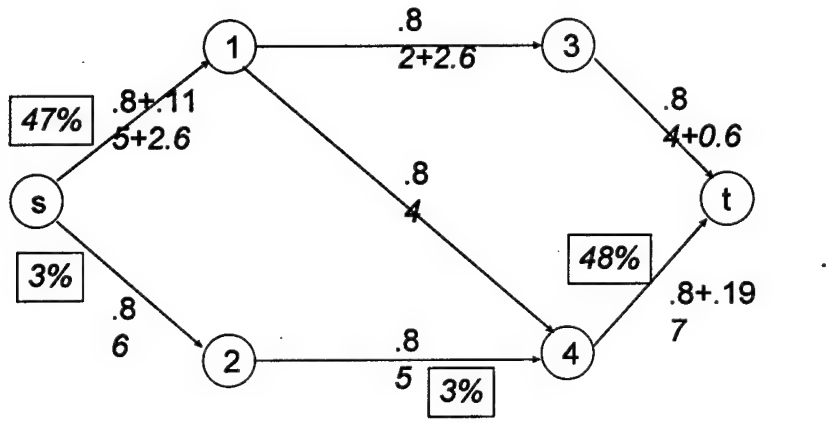


Figure 10. Results from Maximization of Flow Damage Utility with Hardening (Model C).

As anticipated, all three models allocate resources differently. Maximizing an upper bound on expected flow (Model A) is not the same as maximizing an estimate of upper bound on expected flow after a worst case arc loss (Model B). In anticipation of a loss, Model B distributes improvement over more components, whereas Model A, with the exception to its improvement on arc *4-t*, places its resources on a single path—virtually inviting an attack there. The introduction of the hardening game changes decisions even further

(Model C). In Model C, the addition of the hardening game allows the model to allocate hardening effort in addition to component reliability/capacity improvement. Here, the results show that virtually all of the defensive effort should be directed toward making arcs *s-1* and *4-t* difficult to exploit, which makes sense since they are the only two network components that lie on more than one of the paths between source and sink.

Costing Capacity vs. Reliability Improvements

One matter must be addressed that until now has been "swept under the rug"; namely, the issue of a single budget for both capacity and reliability improvements, which implies a conversion factor between the two. An obvious form is this additional constraint added to Models A-D:

$$B_{rel} + \alpha B_{cap} \leq B_{tot}$$

where B_{rel} is the reliability budget, B_{cap} the capacity budget, and B_{tot} the total budget. α coverts B_{cap} into the same units as B_{rel} and B_{tot}. This thesis is not intended to be a treatise on the black art of costing. If the costs of the improvement of both component reliability and capacity, better cost constraints can be generated and used in any of the models described in this research. If the cost relationship between the two improvement types is unknown, the following approach can be used to make some sense of this situation of trying to compare "apples and oranges."

The only place in the MCO model (Model D) which actually forces a trade between capacity and reliability is the flow damage utility criterion function. Reliability damage utility is not a function of any arc capacity constraints.

Even flow damage utility itself is only partially affected by the reliability budget. Assume an infinite reliability improvement budget for a stochastic network. After every component's reliability has been driven to 1, there can be no further increase in flow damage utility. Improvement of the network then becomes a *deterministic* problem—only a function of capacity improvement. In other words, in the MCO problem, it is only when the model forced is to either maximize flow damage utility or meet a flow damage utility target when it compares the efficiencies of component reliability versus capacity improvement. This comparison can, at most, have three possible results: (1) it is more efficient to use component reliability improvement than component capacity improvement; i.e., reliability is relatively cheap; (2) reliability is relatively expensive; and (3) one is roughly as efficient as the other.

In the case of the sample network two graphs are needed to analyze these three possible results: flow damage utility, V_{rel}, versus the budget for reliability improvement, B_{rel}, and versus the budget for capacity improvement, B_{cap}.

Simple linear regression can be used on both graphs (Figures 11 and 12) to determine their approximate slopes. The ratio of these two slopes can be used to calculate the translation value, α, for the case of coarse parity with regard to efficiency.

$$\alpha = \frac{\frac{\Delta V_{rel}}{\Delta B_{cap}}}{\frac{\Delta V_{rel}}{\Delta B_{rel}}} = \frac{\Delta B_{rel}}{\Delta B_{cap}}$$

Vary α by looking at the two other cases, cheap and expensive capacity, by increasing or decreasing it by an order of magnitude. Again, precise numbers are not important here. These three different cost relationships will be used as inputs into Model D, from which respective efficient frontiers may be generated, the analysis of which will be the key to observing the effect of α.

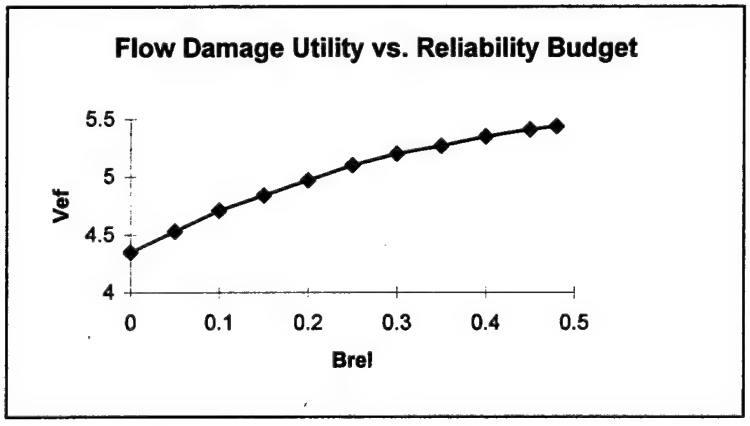


Figure 11. Relationship Between Reliability Improvement Budget and Flow Damage Utility for Sample Network.

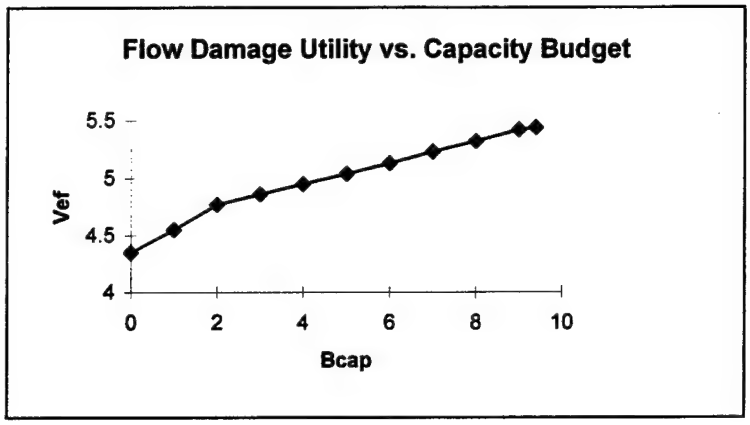


Figure 12. Relationship Between Capacity Improvement Budget and Flow Damage Utility for Sample Network.

Multiple Criteria Optimization—Flow vs. Reliability Damage Utilities

In the MCO Model, an efficient frontier is generated for the two criterion functions, V_{ef} and V_{rel}. In the case of the sample network, three such frontiers were generated, one corresponding to each α (Figures 13-15).

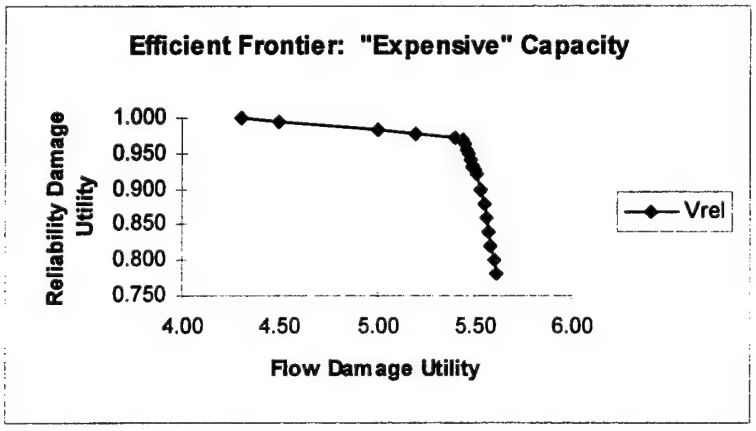


Figure 13. Tradeoff Space Between Flow and Reliability Damage Utilities for Sample Network in the Case of "Expensive" Capacity.

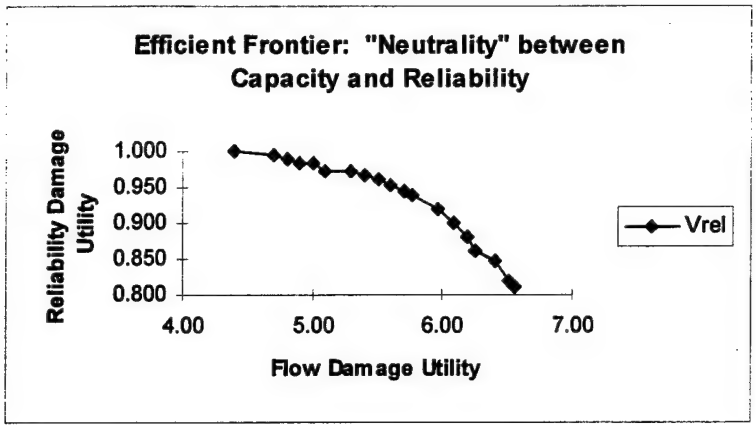


Figure 14. Tradeoff Space Between Flow and Reliability Damage Utilities for Sample Network in the Case of "Neutrality" Between Capacity and Reliability Improvement Costs.

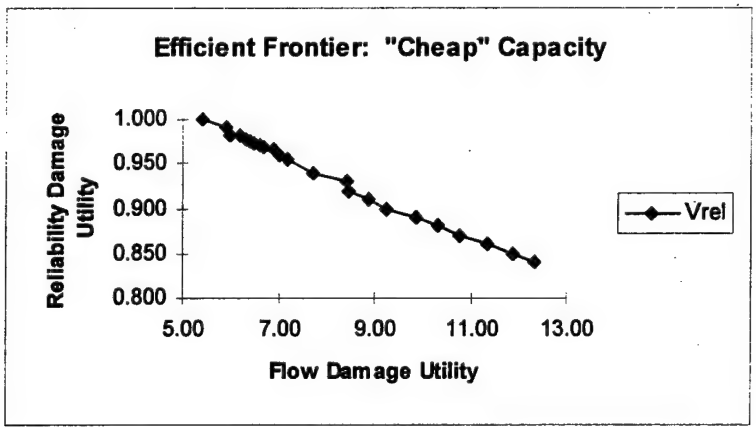


Figure 15. Tradeoff Space Between Flow and Reliability Damage Utilities for Sample Network in the Case of "Cheap" Capacity.

In the case of this network, a clear trend is observed: as capacity improvement becomes cheaper relative to reliability improvement, the efficient frontier flattens. This trend should make intuitive sense. In Figure 13, where reliability is the efficient choice for throughput improvement, there really is not much of a tradeoff between the two metrics until either extreme of the frontier.

The counterexample to Figure 13 is Figure 15, where a clear tradeoff is present. Here, capacity is the efficient choice for V_{ef} improvement—every dollar spent buying more capacity is a dollar that does nothing for V_{rel} improvement.

Analysis of Efficient Frontiers

An advantageous technique used to analyze efficient frontiers is reverse filtering as described by Steuer, the first step of which is to define an ideal point from which to measure deviation [28]. An appropriate ideal point is $\{max\ V_{rel} \mid V_{ef} > 0,\ max\ V_{ef} \mid V_{rel} > 0\}$, an unachievable improvement strategy. Next, after equalizing the ranges of each of the metrics from 0 to 1, the distance from the ideal point (now {1,1}) to any particular improvement strategy is a function of the metric parameter used (Manhattan, Euclidean, or Tchebycheff), each of which carries the implication of a decision maker's tradeoff attitude [7]. *This approach is not classical decision analysis*, where the emphasis is on interaction with a decision maker to ascertain his or her value function for each criterion axis on the efficient frontier. Reverse filtering may be performed *prior to any interaction with a decision maker.*

The L_1 or Manhattan metric corresponds to a totally *compensatory* tradeoff attitude on the part of a decision maker. A compensatory decision maker treats the contributions of each criterion equally. The L_∞ or Tchebycheff metric corresponds to a totally *non-compensatory* tradeoff

attitude. Here, a decision maker is interested in minimizing his or her maximum deviation from the ideal. An intermediate metric, L_2 or Euclidean, is used because of its geometric interpretation. The following equation is the formal representation of L_p-metric calculation under reverse filtering. The term, L_p-metric, is used here as opposed to L_p-norm—they are *not* synonymous. Norm refers to the length of a vector, while metric is used for the distance between two vectors [28].

$$L_p = \left\| \mathbf{v}^t - \mathbf{v}^h \right\|_p^{\pi} = \left[\sum_{i=1}^{q} \left(\pi_i \left| v_i{}^t - v_i{}^h \right| \right)^p \right]^{1/p}$$

q	dimensionality of the vectors being filtered
π_i	range equalization weight
p	metric parameter, $p \in (\{1,2,...\} \cup \{\infty\})$

Figures 16-18 show the results of reverse filtering on each of the three efficient frontiers. The somewhat startling result from this frontier analysis is the existence—in every one of the three cost relationships—of a dominant improvement strategy. Dominant in this context means that a discrete improvement strategy is nearest to the ideal *independent of the metric parameter used.* Since the Manhattan, Euclidean, and Tchebycheff metrics capture the whole spectrum of decision maker tradeoff philosophy, the existence of dominant solutions here means that a "value-free" network improvement strategy has been found in the case of each cost relationship.

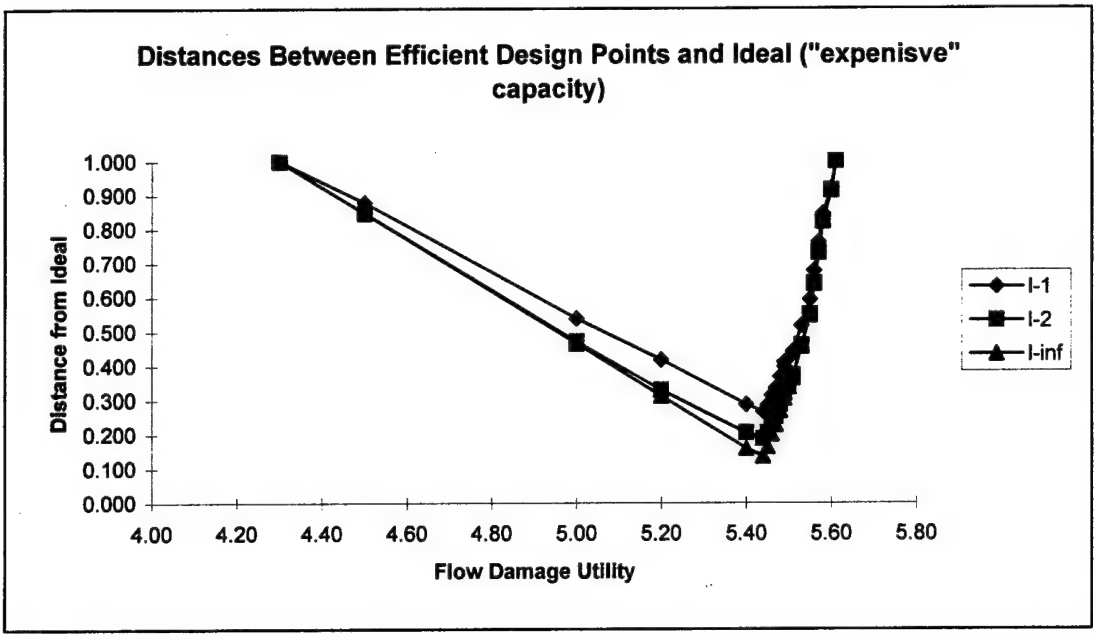


Figure 16.

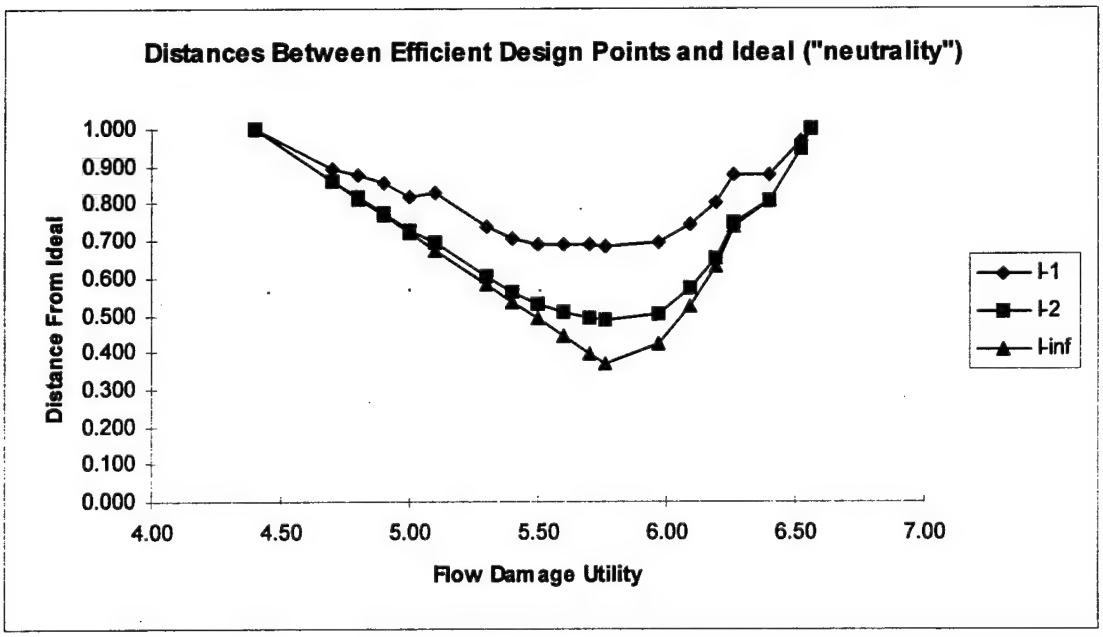


Figure 17.

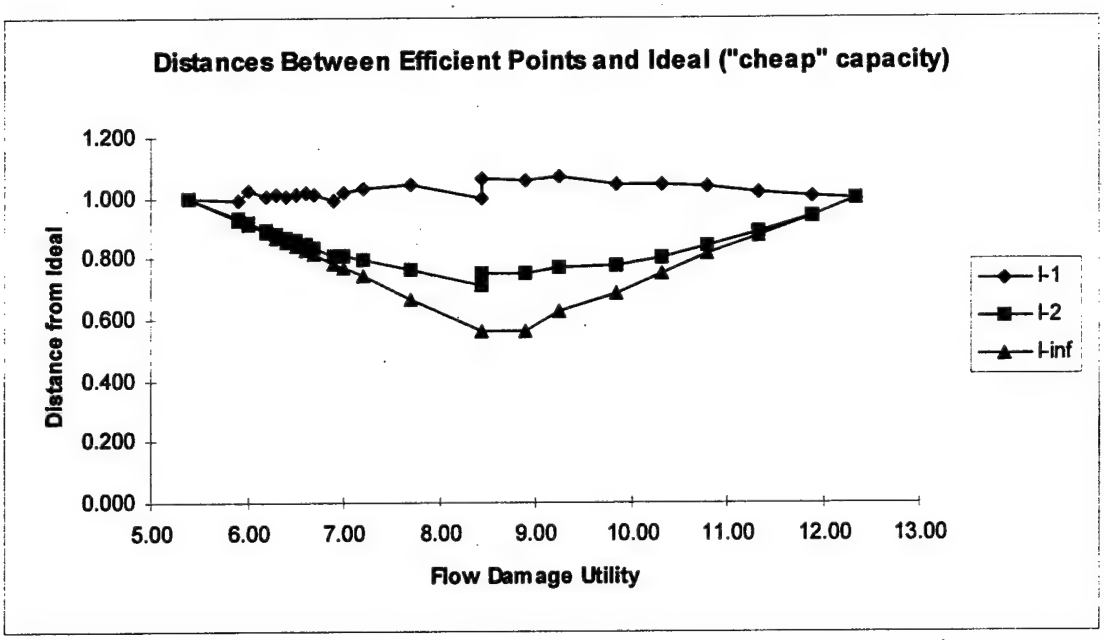


Figure 18.

Value of Hardening

An advantage of using the hardening game in this model of network improvement is in its ability to quantify the value of security. For example, on the sample network, Model C or D can be used as a network improvement model, both of which use the hardening game. To quantify or impute the value of the hardening strategy, run the model again without the hardening game; i.e, set the budget for game variables (y_{ij}s) equal to 0 instead of 1. The difference in budgets is the value of hardening, which can be expressed in terms of reliability improvement, capacity improvement, or, in the case where a conversion factor is used in a total budget constraint, total budget units. Alternatively, after Model C is run, modify Model B to minimize total

budget subject to the flow damage utility calculated by Model C. The difference between the two total budgets is the imputed value of hardening. To illustrate, the sample network and notional budget number are used as follows in Figure 19:

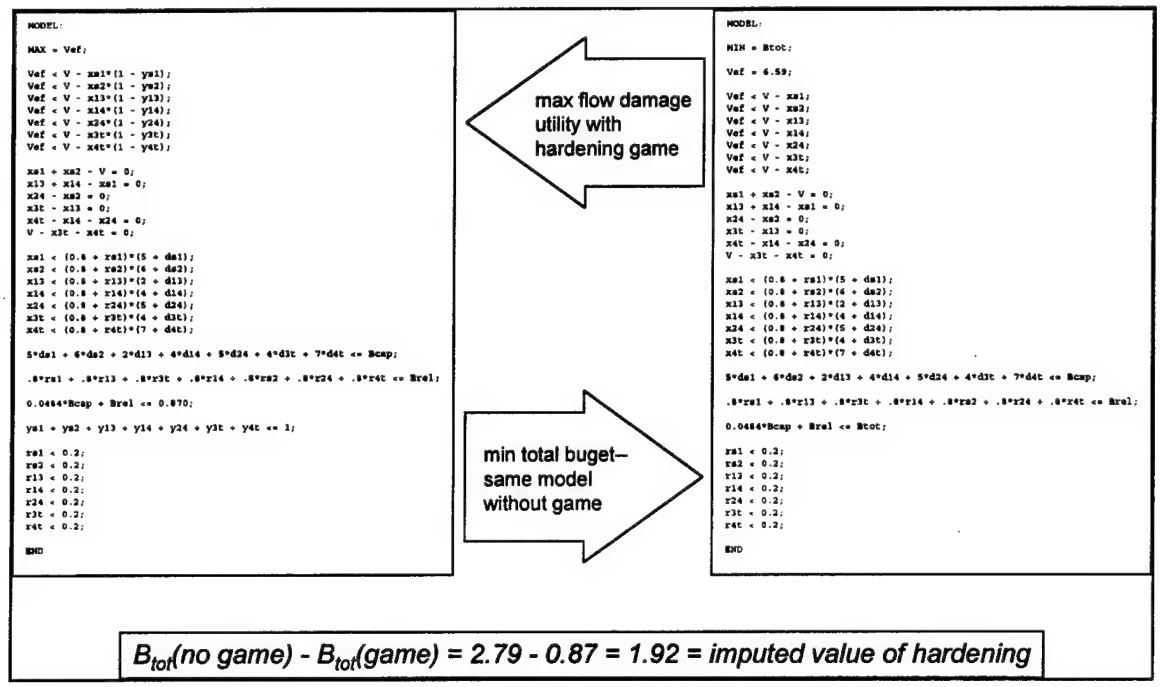


Figure 19. Calculation of the Value of Hardening.

In conventional models like Models 1 and A, it is impossible to quantify the value of making a component more difficult to exploit, because hardness as it has been described in this work does not—at least on appearance—contribute to current operational performance. The situation with these conventional models is similar to quantifying the value of, say, home-owner's insurance: it is only valuable *after* the house has burnt down.

Value of Model without Network Improvement

Another advantage of the described modeling approach is in its ability to produce a hardening (defensive) strategy without running the improvement model. Models 3, 4, C, and D all may be used with a zero improvement budget. This represents a "descriptive" application of the model, where a current performance of a communications network is given. Results from the mathematical program will show not only an optimal hardening strategy, but also an optimal attack strategy. The *shadow prices* (or dual variables at optimality) of the game variables in these models are the best enemy attack strategy. Figures 20 and 21 show the optimal defensive/offensive strategy for the sample network in the case of a zero improvement budget.

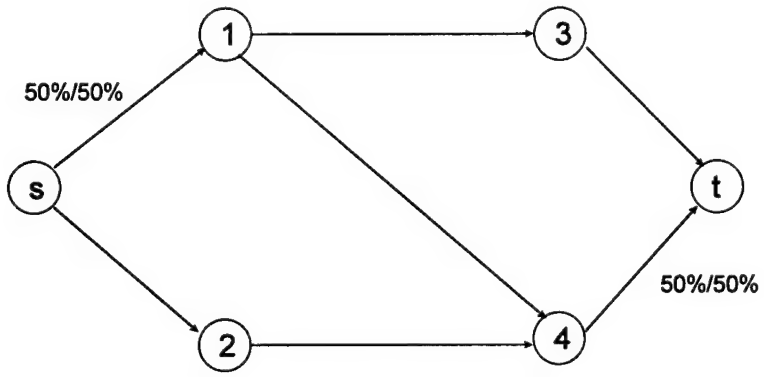


Figure 20. V_{rel}-based Hardening Defensive/Offensive Strategy for Sample Network without Improvement Budget (Model 3).

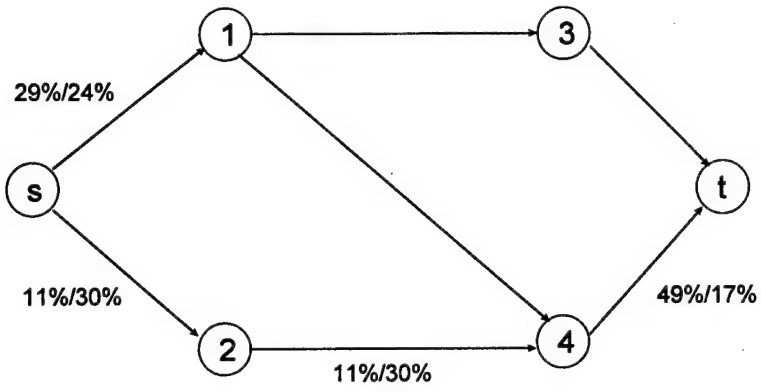


Figure 21. V_{ef}-based Hardening Defensive/Offensive Strategy for Sample Network without Improvement Budget (Model C).

In Figure 20, because the metric is reliability-based, both defense and offense concentration on nodes *s-1* and *4-t* make sense: they are the only two that lie on more than one path from source to sink. This symmetry, however, disappears in Figure 21, where the flow metric is considered. Here, it is perhaps more visible that the two strategies are generated from a game-theoretic model that takes into consideration the moves of one's adversary. Where as Blue's defensive effort remains attached to the two "path-heavy" arcs, *s-1* and *4-t*, Red's counter effort is primarily focused on arcs *s-2* and *2-4*.

V. Work on Larger Networks

Introduction

This thesis is part methodology, part application. Application of the models and concepts discussed in the Chapters I-IV to larger, more realistic networks of interest to the Department of Defense is investigated here in this chapter. Appendix B contains LINGO sample computations from this chapter.

Computational Considerations

Three major computational difficulties arise when networks of this size, geometry, and complexity are investigated: constraint generation, calculation of symbolic expressions for two-terminal reliability (both the R_{st} and R_{ij} terms), and solver capability when running models with large numbers of highly nonlinear, complicated constraints. For illustration, the reader is invited to compare the MCO program (Model D) of the sample network (Appendix A) to that of Network A (Appendix B). The complexity of the NLP for Network A, composed of 15 nodes and 21 arcs, is several pages and includes large, highly-nonlinear terms. (Inspect just one of the R_{ij} expressions.) By comparison, the NLP for the sample network, composed of 6 nodes and 7 arcs, is quite tractable. Its most complicated term, the R_{st}

expression, contains 7 terms, the highest order of which is the product of all seven arc operational probabilities.

The first problem, constraint generation, is the easy one to solve. LINGO and similar products will, when given instructions in their own scripting language, generate constraints automatically [20]. In the case of Model C, LINGO requires only the input of node/arc information to generate all of the constraints. (See Appendix B for examples.)

The second problem, calculation of reliability expressions, is discussed in detail in Appendix C. Regardless of the method used (two are discussed in this thesis, factoring and an algebraic approach), R_{st} calculation involves an inefficient (non-polynomial) algorithm. For networks of the size of A, B, and C, calculation of reliability expressions alone is a formidable problem, which leads directly to the third problem of applying these models to large networks.

The third problem is solver capability. Although reliability expressions can (and have been) generated for Network B (beyond Network A), HyperLingo will not find feasible solutions due to the size of the problem. Model D—the bicriteria model—was applied to Network A, and the results from that work are presented in this chapter. Applying Model D to Network C was not attempted (since it is larger than Network B, which is too big for the solver). Results, however, were obtained from applying Model C—the flow damage utility game—to all three networks, A, B, and C.

Networks A, B, and C

Networks A, B, and C are representative of larger, more realistic information networks of interest to the Department of Defense. Network A is shown in Figure 22. In Network A, note that all paths travel through node *14*. Network B is shown in Figure 23. In Network B, although node *16* appears to be critical, numerous other paths exist from source to sink. Network C is shown in Figure 24. In Network C, node *22* contains all paths from source to sink.

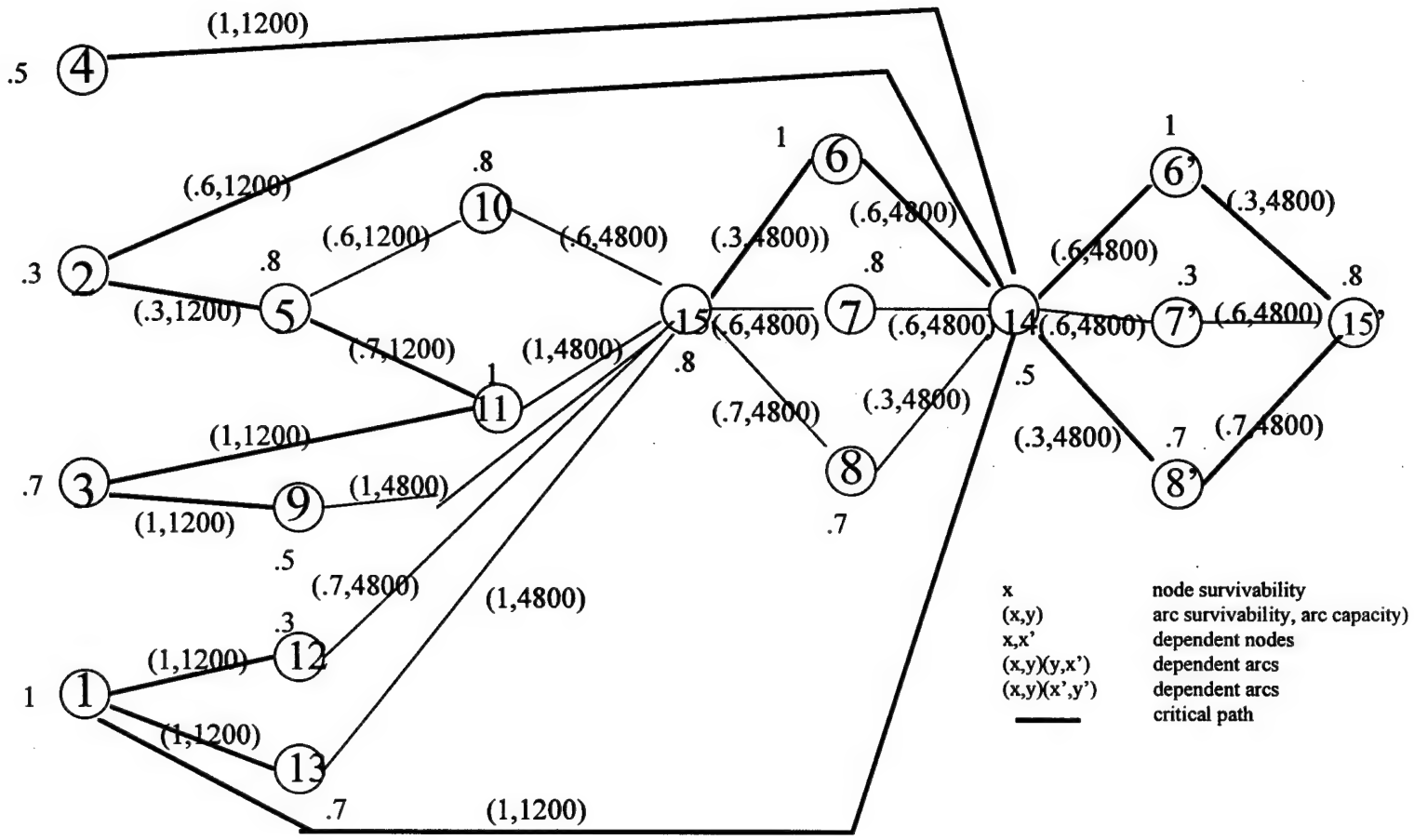


Figure 22. Network A.

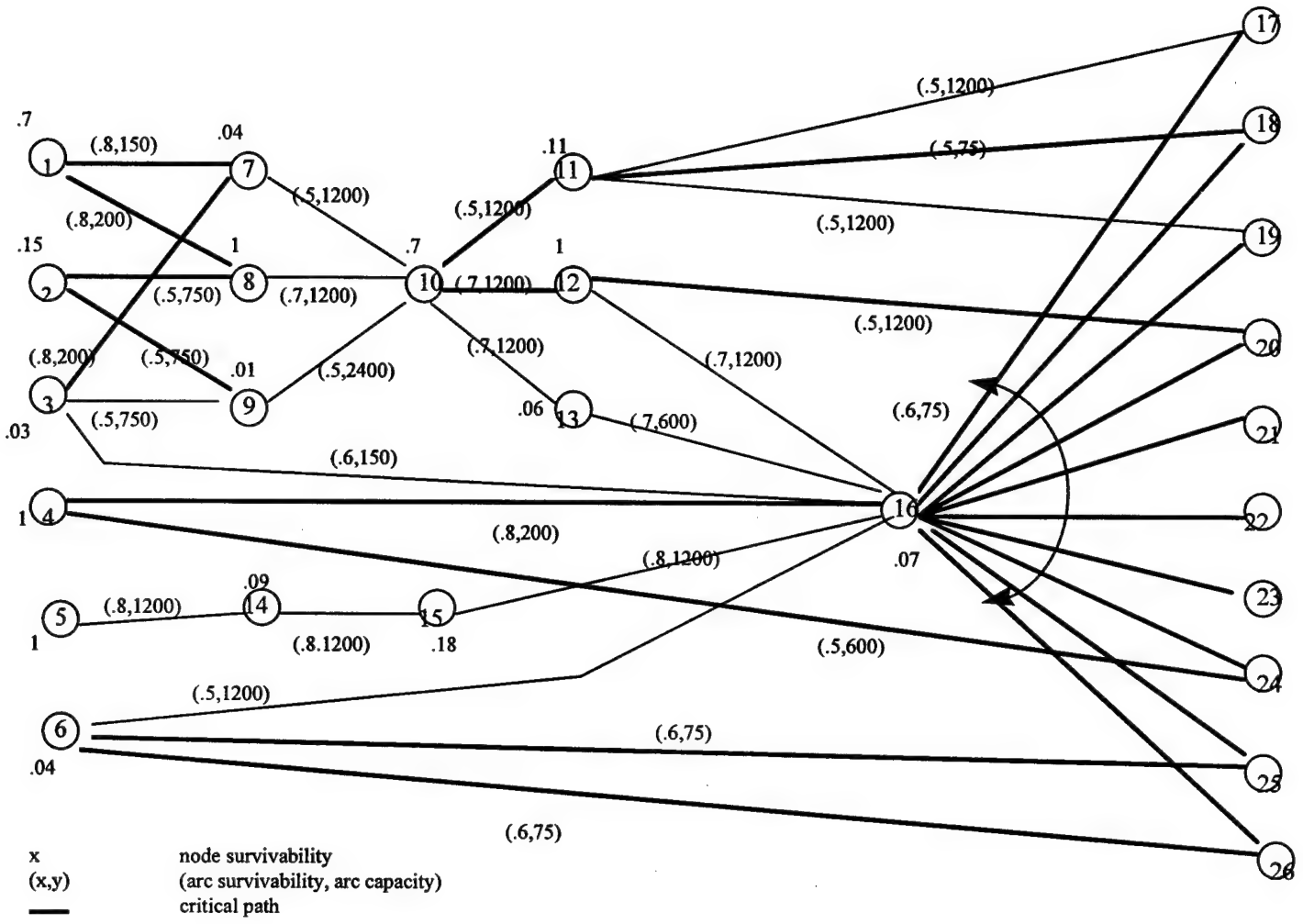


Figure 23. Network B.

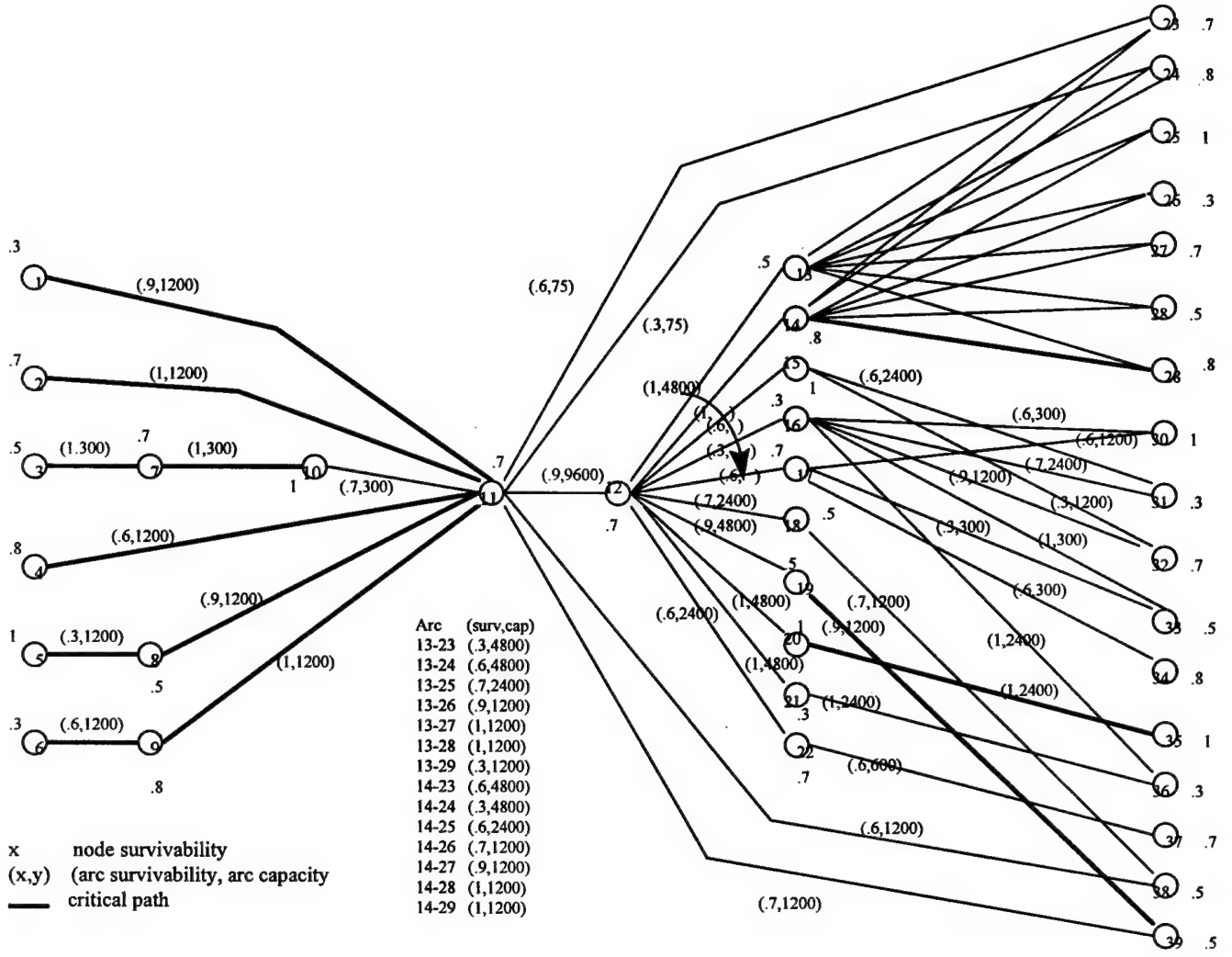


Figure 24. Network C.

Model C Applied to Networks A, B, and C

Tables 1-3 show the results from the application of Model C to the three networks under investigation. From left to right: node reliability improvement (r_k), arc reliability improvement (r_{ij}), arc capacity improvement (d_{ij}), defensive game variable (y_{ij}), and offensive game variable (z_{ij}).

component	r_k	r_{ij}	d_{ij}	y_{ij}	z_{ij}
8	0.02				
15	0.04				
(4,14)			833	0.17	0.36
(6,14)					0.26
(7,14)					0.11
(8,14)		0.05		0.41	0.02
(15,6)		0.03		0.41	0.26

Table 1. Network A Results (Model C).

In Network A, Blue's strategy centers around hardening and improving the reliability of arcs *8-14* and *15-6*, although other components are improved. Red's strategy is also mixed, with more than a third of its attack effort being devoted to arc *4-14*.

component	r_k	r_{ij}	d_{ij}	y_{ij}	z_{ij}
10	0.20				
(6,26)			67		
(8,10)				0.30	0.35
(10,12)				0.38	0.31
(12,20)				0.32	0.34

Table 2. Network B Results (Model C).

In Network B, perhaps counter-intuitively, node *16* is *not* improved. The Blue-Red hardening/attacking strategies center around the trunk *8-10-12-20*.

component	r_k	r_{ij}	d_{ij}	y_{ij}	z_{ij}
11	.14				
(1,11)				0.09	0.46
(2,11)				0.18	0.41
(3,7)			589		
(7,10)			322		
(10,11)			755		
(11,12)				0.73	0.13

Table 3. Network C Results (Model C).

In Network C, another perhaps counter-intuitive result is observed. The "obvious" attack strategy, namely, all offensive resources against arc *11-12*, is not pursued. Rather, Red anticipates Blue's defensive strategy and acts accordingly.

Model D Applied to Network A

The MCO model, Model D, was applied to Network A with results similar to those from the sample network. Reverse filtering shows the existence of a dominating improvement strategy. Figure 25 presents the tradeoff space between the reliability and flow game-theoretic metrics, and Figure 26 contains the results from reverse filtering as presented in Chapter IV.

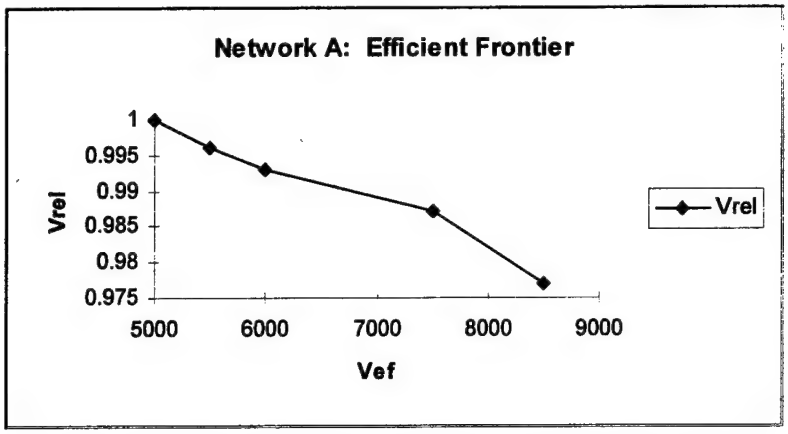


Figure 25. Tradeoff Space Between Flow and Reliability Damage Utilities for Network A.

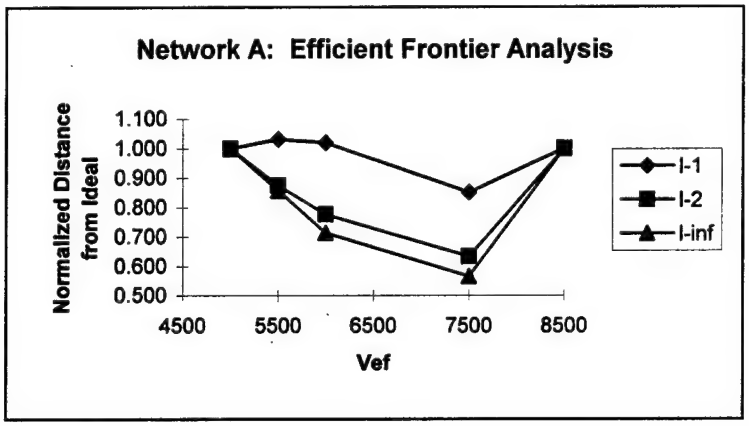


Figure 26. Reverse Filtering Results for Network A.

Like the case of the sample network in Chapter IV, the "kink" in the efficient frontier here is exploited by reverse filtering which reveals a "value-free" improvement strategy. It is emphasized here that this thesis does not contain a *proof* that a "value-free" solution must exist for all conceivable networks—future work in this field may lead to an understanding of what set of conditions is necessary for one to be present. It is encouraging, however,

that two *very different* networks, the sample network and Network A, display this property.

VI. Conclusions and Recommendations

Summary

This thesis proposed a new, game-theoretic, network flow criterion, V_{ef}, for network improvement and security. This new measure was demonstrated in a multicriteria optimization model which used it with a related, previously developed, reliability-based metric, V_{rel}.

Conclusions

- The multiple criteria game, Model D, capturing the strategic competition between network defender and attacker, yields a "value-free" solution—independent of decision maker tradeoff philosophy—for the networks investigated.
- Irrespective of unit costs of reliability vs. bandwidth improvement, a "value-free" solution may be obtained. Cumbersome cost issues are therefore removed from the debate over optimal defense/performance strategy until "hard" numbers become available.
- The value of information security may be imputed from these game-theoretic models.
- V_{ef} and V, the expected throughput, are different measures of effectiveness. A network optimized for an upper bound on expected

flow is not optimized for that upper bound after a worst-case component loss estimate.

- Shadow prices of the game constraints in Models C and D reveal optimal enemy attack strategy.

Future Work

- Computational issues remain with both the generation of reliability expressions for larger networks and the use of these expressions in nonlinear solvers. A large scale systems optimization approach, which might feature some form of decomposition, appears to be necessary for analysis of networks larger than Network A. Specialized nonlinear solvers running on computer hardware larger than either personal computers or perhaps even engineering workstations will be helpful at the minimum, if not absolutely necessary, to this end.
- This thesis emphasized the "game" half of its "game-theoretic" modeling. The "theory" portion should be pursued more rigorously to understand both the true nature of the equilibrium solution found in Models C (single criteria flow) and D (multiple criteria) and the implications of the respective payoff tables' being composed of variables. The author has argued, with provided justification, that the network hardening game is, in fact, a two-person, zero-

sum, non-cooperative game by commonly accepted definitions in the literature. Interaction with up-to-date game theoreticians should answer the question, "Is the hardening game *more* than this, on account of the optimization problem underlying its payoff matrix?"

- Further research into the conditions under which "value-free" solutions occur in Model D, the MCO model, should be conducted. Although this thesis demonstrated a "value-free" outcome for two different networks, there is no *guarantee* as yet that such an outcome is possible for *all* networks.
- After Lyle's work on a reliability-based MOE and this thesis' investigation of a flow-based metric and the combined use of both metrics in an MCO model, this area is ripe for a purely applications-oriented research project, in which a specific physical system is modeled for improvement and defense.

Appendix A: LINGO Input/Output Files for Sample Network

Appendix A contains input and output examples using the sample network.

It includes, in order:

- Model 1
- Model 2
- Model 3
- Model 4
- Model A
- Model B
- Model C
- Model D (including tables with the data for all three efficient frontier calculations)

```
! Model 1 maximizes the two-terminal reliability of the sample network;

MODEL:

MAX = Rst;

! symbolic expression for two-terminal reliability;
Rst = (.8 + rs1)*(.8 + r13)*(.8 + r3t)
+ (.8 + rs1)*(.8 + r14)*(.8 + r4t)
+ (.8 + rs2)*(.8 + r24)*(.8 + r4t)
- (.8 + rs1)*(.8 + r13)*(.8 + r3t)*(.8 + r14)*(.8 + r4t)
- (.8 + rs1)*(.8 + r14)*(.8 + rs2)*(.8 + r24)*(.8 + r4t)
- (.8 + rs1)*(.8 + r13)*(.8 + r3t)*(.8 + rs2)*(.8 + r24)*(.8 + r4t)
+ (.8 + rs1)*(.8 + r13)*(.8 + r3t)*(.8 + r14)*(.8 + rs2)*(.8 + r24)
*(.8 + r4t);

! improvement budget for component reliability improvement;
.8*rs1 + .8*r13 + .8*r3t + .8*r14 + .8*rs2 + .8*r24 + .8*r4t <= 0.48;

! maximum component reliability is 1.
rs1 < 0.2;
rs2 < 0.2;
r13 < 0.2;
r14 < 0.2;
r24 < 0.2;
r3t < 0.2;
r4t < 0.2;

END
```

OUTPUT FOR SAMPLE NETWORK: MODEL 1

Rows= 10 Vars= 8 No. integer vars= 0
Nonlinear rows= 1 Nonlinear vars= 7 Nonlinear constraints= 1
Nonzeros= 32 Constraint nonz= 22 Density=0.356

Optimal solution found at step: 8
Objective value: 1.000000

Variable	Value	Reduced Cost
RST	1.000000	0.0000000E+00
RS1	0.2000000	0.0000000E+00
R13	0.0000000E+00	0.2220446E-07
R3T	0.0000000E+00	0.2220446E-07
R14	0.2000000	0.0000000E+00
R4T	0.2000000	0.0000000E+00
RS2	0.2499936E-08	0.0000000E+00
R24	0.0000000E+00	0.0000000E+00

Row	Slack or Surplus	Dual Price
1	1.000000	1.000000
2	-0.1149625E-07	1.000000
3	0.0000000E+00	0.0000000E+00
4	0.0000000E+00	0.3600000
5	0.2000000	0.0000000E+00
6	0.2000000	0.0000000E+00
7	0.0000000E+00	0.1296000
8	0.2000000	0.0000000E+00
9	0.2000000	0.0000000E+00
10	0.0000000E+00	0.3600000

```
! Model 2 maximizes the reliability damage utility of the sample
network;

MODEL:

MAX = Vrel;

Rst = (.8 + rs1)*(.8 + r13)*(.8 + r3t)
+ (.8 + rs1)*(.8 + r14)*(.8 + r4t)
+ (.8 + rs2)*(.8 + r24)*(.8 + r4t)
- (.8 + rs1)*(.8 + r13)*(.8 + r3t)*(.8 + r14)*(.8 + r4t)
- (.8 + rs1)*(.8 + r14)*(.8 + rs2)*(.8 + r24)*(.8 + r4t)
- (.8 + rs1)*(.8 + r13)*(.8 + r3t)*(.8 + rs2)*(.8 + r24)*(.8 + r4t)
+ (.8 + rs1)*(.8 + r13)*(.8 + r3t)*(.8 + r14)*(.8 + rs2)*(.8 + r24)
*(.8 + r4t);

! expression for two-terminal reliability when arc s-1 is destroyed;
RA = (.8 + rs2)*(.8 + r24)*(.8 + r4t);

! expression for two-terminal reliability when arc 1-3 is destroyed;
RB = (.8 + rs1)*(.8 + r14)*(.8 + r4t)
+ (.8 + rs2)*(.8 + r24)*(.8 + r4t)
- (.8 + rs1)*(.8 + r14)*(.8 + rs2)*(.8 + r24)*(.8 + r4t);

! expression for two-terminal reliability when arc 3-t is destroyed;
RC = RB;

! expression for two-terminal reliability when arc 1-4 is destroyed;
RD = (.8 + rs1)*(.8 + r13)*(.8 + r3t)
+ (.8 + rs2)*(.8 + r24)*(.8 + r4t)
- (.8 + rs1)*(.8 + r13)*(.8 + r3t)*(.8 + rs2)*(.8 + r24)*(.8 + r4t);

! expression for two-terminal reliability when arc s-2 is destroyed;
RE = (.8 + rs1)*(.8 + r13)*(.8 + r3t)
+ (.8 + rs1)*(.8 + r14)*(.8 + r4t)
- (.8 + rs1)*(.8 + r13)*(.8 + r3t)*(.8 + r14)*(.8 + r4t);

! expression for two-terminal reliability when arc 2-4 is destroyed;
RF = RE;

! expression for two-terminal reliability when arc 4-t is destroyed;
RG = (.8 + rs1)*(.8 + r13)*(.8 + r3t);

! value function for reliability damage utility;
Vrel < (RA - Rst + 1);
Vrel < (RB - Rst + 1);
Vrel < (RC - Rst + 1);
Vrel < (RD - Rst + 1);
Vrel < (RE - Rst + 1);
Vrel < (RF - Rst + 1);
Vrel < (RG - Rst + 1);
```

```
.8*rs1 + .8*r13 + .8*r3t + .8*r14 + .8*rs2 + .8*r24 + .8*r4t <= 0.48;

rs1 < 0.2;
rs2 < 0.2;
r13 < 0.2;
r14 < 0.2;
r24 < 0.2;
r3t < 0.2;
r4t < 0.2;

END
```

OUTPUT FOR SAMPLE NETWORK: MODEL 2

Rows= 24 Vars= 16 No. integer vars= 0
Nonlinear rows= 6 Nonlinear vars= 7 Nonlinear constraints= 6
Nonzeros= 96 Constraint nonz= 74 Density=0.235

Optimal solution found at step: 10
Objective value: 0.7944168

Variable	Value	Reduced Cost
VREL	0.7944168	0.0000000E+00
RST	0.9275832	0.0000000E+00
RS1	0.0000000E+00	0.2187209E-01
R13	0.1499980	0.0000000E+00
R3T	0.1500020	0.0000000E+00
R14	0.0000000E+00	0.2222300
R4T	0.0000000E+00	0.2187793E-01
RS2	0.1500196	0.0000000E+00
R24	0.1499804	0.0000000E+00
RA	0.7220000	0.0000000E+00
RB	0.7719200	0.0000000E+00
RC	0.7719200	0.0000000E+00
RD	0.9227160	0.0000000E+00
RE	0.7719200	0.0000000E+00
RF	0.7719200	0.0000000E+00
RG	0.7220000	0.0000000E+00

Row	Slack or Surplus	Dual Price
1	0.7944168	1.000000
2	0.0000000E+00	-1.000000
3	0.0000000E+00	0.4999968
4	0.0000000E+00	0.0000000E+00
5	0.0000000E+00	0.0000000E+00
6	0.0000000E+00	0.0000000E+00
7	0.0000000E+00	0.0000000E+00
8	0.0000000E+00	0.0000000E+00
9	0.0000000E+00	0.5000032
10	0.0000000E+00	0.4999968
11	0.4992000E-01	0.0000000E+00
12	0.4992000E-01	0.0000000E+00
13	0.2007160	0.0000000E+00
14	0.4992000E-01	0.0000000E+00
15	0.4992000E-01	0.0000000E+00
16	0.0000000E+00	0.5000032
17	0.1072884E-07	0.2701825
18	0.2000000	0.0000000E+00
19	0.4998045E-01	-0.8894947E-05
20	0.5000203E-01	0.9157774E-06
21	0.2000000	0.0000000E+00
22	0.5001956E-01	0.0000000E+00

23	0.4999798E-01	0.0000000E+00
24	0.2000000	0.0000000E+00

```
! Model 3 maximizes the reliability damage utility of the sample network
with the hardening game;

MODEL:

MAX = Vrel;

Rst = (.8 + rs1)*(.8 + r13)*(.8 + r3t)
+ (.8 + rs1)*(.8 + r14)*(.8 + r4t)
+ (.8 + rs2)*(.8 + r24)*(.8 + r4t)
- (.8 + rs1)*(.8 + r13)*(.8 + r3t)*(.8 + r14)*(.8 + r4t)
- (.8 + rs1)*(.8 + r14)*(.8 + rs2)*(.8 + r24)*(.8 + r4t)
- (.8 + rs1)*(.8 + r13)*(.8 + r3t)*(.8 + rs2)*(.8 + r24)*(.8 + r4t)
+ (.8 + rs1)*(.8 + r13)*(.8 + r3t)*(.8 + r14)*(.8 + rs2)*(.8 + r24)
*(.8 + r4t);

! expression for two-terminal reliability when arc s-1 is destroyed;
RA = (.8 + rs2)*(.8 + r24)*(.8 + r4t);

! expression for two-terminal reliability when arc 1-3 is destroyed;
RB = (.8 + rs1)*(.8 + r14)*(.8 + r4t)
+ (.8 + rs2)*(.8 + r24)*(.8 + r4t)
- (.8 + rs1)*(.8 + r14)*(.8 + rs2)*(.8 + r24)*(.8 + r4t);

! expression for two-terminal reliability when arc 3-t is destroyed;
RC = RB;

! expression for two-terminal reliability when arc 1-4 is destroyed;
RD = (.8 + rs1)*(.8 + r13)*(.8 + r3t)
+ (.8 + rs2)*(.8 + r24)*(.8 + r4t)
- (.8 + rs1)*(.8 + r13)*(.8 + r3t)*(.8 + rs2)*(.8 + r24)*(.8 + r4t);

! expression for two-terminal reliability when arc s-2 is destroyed;
RE = (.8 + rs1)*(.8 + r13)*(.8 + r3t)
+ (.8 + rs1)*(.8 + r14)*(.8 + r4t)
- (.8 + rs1)*(.8 + r13)*(.8 + r3t)*(.8 + r14)*(.8 + r4t);

! expression for two-terminal reliability when arc 2-4 is destroyed;
RF = RE;

! expression for two-terminal reliability when arc 4-t is destroyed;
RG = (.8 + rs1)*(.8 + r13)*(.8 + r3t);

! value function for reliability damage utility;
Vrel < (RA - Rst + 1)*(1 - ys1) + ys1;
Vrel < (RB - Rst + 1)*(1 - y13) + y13;
Vrel < (RC - Rst + 1)*(1 - y3t) + y3t;
Vrel < (RD - Rst + 1)*(1 - y14) + y14;
Vrel < (RE - Rst + 1)*(1 - ys2) + ys2;
Vrel < (RF - Rst + 1)*(1 - y24) + y24;
Vrel < (RG - Rst + 1)*(1 - y4t) + y4t;
```

```
ys1 + ys2 + y13 + y14 + y24 + y3t + y4t <= 1;

.8*rs1 + .8*r13 + .8*r3t + .8*r14 + .8*rs2 + .8*r24 + .8*r4t <= 0.48;

rs1 < 0.2;
rs2 < 0.2;
r13 < 0.2;
r14 < 0.2;
r24 < 0.2;
r3t < 0.2;
r4t < 0.2;

END
```

OUTPUT FOR SAMPLE PROBLEM: MODEL 3

```
Rows=      25 Vars=      23 No. integer vars=       0
 Nonlinear rows=      13 Nonlinear vars=      22 Nonlinear constraints=
13
 Nonzeros=     111 Constraint nonz=     88 Density=0.185

 Optimal solution found at step:        15
 Objective value:                0.9164684
```

Variable	Value	Reduced Cost
VREL	0.9164684	0.0000000E+00
RST	0.9480759	0.0000000E+00
RS1	0.1699061	0.0000000E+00
R13	0.0000000E+00	0.1984139
R3T	0.0000000E+00	0.1984140
R14	0.0000000E+00	0.1262027
R4T	0.7280912E-01	0.0000000E+00
RS2	0.1786876	0.0000000E+00
R24	0.1785971	0.0000000E+00
RA	0.8359250	0.0000000E+00
RB	0.8645443	0.0000000E+00
RC	0.8645443	0.0000000E+00
RD	0.9377729	0.0000000E+00
RE	0.8645443	0.0000000E+00
RF	0.8645443	0.0000000E+00
RG	0.6207399	0.0000000E+00
YS1	0.2551860	0.0000000E+00
Y13	0.0000000E+00	0.4561578E-01
Y3T	0.0000000E+00	0.3414669E-01
Y14	0.0000000E+00	0.4561578E-01
YS2	0.0000000E+00	0.1916911E-01
Y24	0.0000000E+00	0.4561578E-01
Y4T	0.7448140	0.0000000E+00

Row	Slack or Surplus	Dual Price
1	0.9164684	1.000000
2	0.0000000E+00	-0.7924134
3	0.0000000E+00	0.3029425
4	0.0000000E+00	0.1373025
5	0.0000000E+00	0.1373025
6	0.0000000E+00	0.0000000E+00
7	0.0000000E+00	0.3166070
8	0.0000000E+00	0.0000000E+00
9	0.0000000E+00	0.3556136E-01
10	0.0000000E+00	0.4067358
11	0.0000000E+00	0.0000000E+00
12	0.0000000E+00	0.1373025
13	0.7322859E-01	0.0000000E+00
14	0.0000000E+00	0.3166070
15	0.0000000E+00	0.0000000E+00

16	0.0000000E+00	0.1393547
17	0.0000000E+00	0.4561578E-01
18	0.1072884E-07	0.2717463
19	0.3009385E-01	0.0000000E+00
20	0.2131239E-01	0.0000000E+00
21	0.2000000	0.0000000E+00
22	0.2000000	0.0000000E+00
23	0.2140290E-01	0.2008313E-04
24	0.2000000	0.0000000E+00
25	0.1271909	0.0000000E+00

```
! Model 4 maximizes the reliability damage utility of the sample network
with the hardening game;

MODEL:

MAX = Vrel;

Rst >= 0.975;

Rst = (.8 + rs1)*(.8 + r13)*(.8 + r3t)
+ (.8 + rs1)*(.8 + r14)*(.8 + r4t)
+ (.8 + rs2)*(.8 + r24)*(.8 + r4t)
- (.8 + rs1)*(.8 + r13)*(.8 + r3t)*(.8 + r14)*(.8 + r4t)
- (.8 + rs1)*(.8 + r14)*(.8 + rs2)*(.8 + r24)*(.8 + r4t)
- (.8 + rs1)*(.8 + r13)*(.8 + r3t)*(.8 + rs2)*(.8 + r24)*(.8 + r4t)
+ (.8 + rs1)*(.8 + r13)*(.8 + r3t)*(.8 + r14)*(.8 + rs2)*(.8 + r24)
*(.8 + r4t);

! expression for two-terminal reliability when arc s-1 is destroyed;
RA = (.8 + rs2)*(.8 + r24)*(.8 + r4t);

! expression for two-terminal reliability when arc 1-3 is destroyed;
RB = (.8 + rs1)*(.8 + r14)*(.8 + r4t)
+ (.8 + rs2)*(.8 + r24)*(.8 + r4t)
- (.8 + rs1)*(.8 + r14)*(.8 + rs2)*(.8 + r24)*(.8 + r4t);

! expression for two-terminal reliability when arc 3-t is destroyed;
RC = RB;

! expression for two-terminal reliability when arc 1-4 is destroyed;
RD = (.8 + rs1)*(.8 + r13)*(.8 + r3t)
+ (.8 + rs2)*(.8 + r24)*(.8 + r4t)
- (.8 + rs1)*(.8 + r13)*(.8 + r3t)*(.8 + rs2)*(.8 + r24)*(.8 + r4t);

! expression for two-terminal reliability when arc s-2 is destroyed;
RE = (.8 + rs1)*(.8 + r13)*(.8 + r3t)
+ (.8 + rs1)*(.8 + r14)*(.8 + r4t)
- (.8 + rs1)*(.8 + r13)*(.8 + r3t)*(.8 + r14)*(.8 + r4t);

! expression for two-terminal reliability when arc 2-4 is destroyed;
RF = RE;

! expression for two-terminal reliability when arc 4-t is destroyed;
RG = (.8 + rs1)*(.8 + r13)*(.8 + r3t);

! value function for reliability damage utility;
Vrel < (RA - Rst + 1)*(1 - ys1) + ys1;
Vrel < (RB - Rst + 1)*(1 - y13) + y13;
Vrel < (RC - Rst + 1)*(1 - y3t) + y3t;
Vrel < (RD - Rst + 1)*(1 - y14) + y14;
Vrel < (RE - Rst + 1)*(1 - ys2) + ys2;
Vrel < (RF - Rst + 1)*(1 - y24) + y24;
```

```
Vrel < (RG - Rst + 1)*(1 - y4t) + y4t;

ys1 + ys2 + y13 + y14 + y24 + y3t + y4t <= 1;

.8*rs1 + .8*r13 + .8*r3t + .8*r14 + .8*rs2 + .8*r24 + .8*r4t <= 0.48;

rs1 < 0.2;
rs2 < 0.2;
r13 < 0.2;
r14 < 0.2;
r24 < 0.2;
r3t < 0.2;
r4t < 0.2;

END
```

OUTPUT FOR SAMPLE NETWORK: MODEL 4

```
Rows=      26 Vars=      23 No. integer vars=       0
 Nonlinear rows=      13 Nonlinear vars=      22 Nonlinear constraints=
13
 Nonzeros=     113 Constraint nonz=     89 Density=0.181

 Optimal solution found at step:        10
 Objective value:                0.9065511
```

Variable	Value	Reduced Cost
VREL	0.9065511	0.0000000E+00
RST	0.9750000	0.0000000E+00
RS1	0.1696861	0.0000000E+00
R13	0.1356726	0.0000000E+00
R3T	0.1356721	0.0000000E+00
R14	0.0000000E+00	0.1991578
R4T	0.1589691	0.0000000E+00
RS2	0.0000000E+00	0.2202840
R24	0.0000000E+00	0.2202840
RA	0.6137402	0.0000000E+00
RB	0.8815512	0.0000000E+00
RC	0.8815512	0.0000000E+00
RD	0.9416530	0.0000000E+00
RE	0.9415743	0.0000000E+00
RF	0.9415743	0.0000000E+00
RG	0.8489435	0.0000000E+00
YS1	0.7413250	0.0000000E+00
Y13	0.0000000E+00	0.3020844E-01
Y3T	0.0000000E+00	0.6182865E-01
Y14	0.0000000E+00	0.6182865E-01
YS2	0.0000000E+00	0.6182865E-01
Y24	0.0000000E+00	0.6182865E-01
Y4T	0.2586750	0.0000000E+00

Row	Slack or Surplus	Dual Price
1	0.9065511	1.000000
2	0.2384186E-07	-0.4459707
3	0.0000000E+00	-0.3002777
4	0.0000000E+00	0.4427153E-01
5	0.0000000E+00	0.3383691
6	0.0000000E+00	0.0000000E+00
7	0.0000000E+00	0.0000000E+00
8	0.0000000E+00	0.0000000E+00
9	0.0000000E+00	0.0000000E+00
10	0.0000000E+00	0.3636077
11	0.0000000E+00	0.1711473
12	0.0000000E+00	0.3383691
13	0.0000000E+00	0.0000000E+00
14	0.6010179E-01	0.0000000E+00
15	0.6002309E-01	0.0000000E+00

16	0.6002309E-01	0.0000000E+00
17	0.0000000E+00	0.4904836
18	0.0000000E+00	0.6182865E-01
19	0.1072884E-07	0.3748934
20	0.3031389E-01	0.0000000E+00
21	0.2000000	0.0000000E+00
22	0.6432738E-01	-0.1632672E-06
23	0.2000000	0.0000000E+00
24	0.2000000	0.0000000E+00
25	0.6432788E-01	0.0000000E+00
26	0.4103087E-01	0.0000000E+00

```
! Model A maximizes an upper bound of expected throughput.;

MODEL:

MAX = V;

! flow balance equations;
xs1 + xs2 - V = 0;
x13 + x14 - xs1 = 0;
x24 - xs2 = 0;
x3t - x13 = 0;
x4t - x14 - x24 = 0;
V - x3t - x4t = 0;

! arc capacity bounds;
xs1 < (0.8 + rs1)*(5 + ds1);
xs2 < (0.8 + rs2)*(6 + ds2);
x13 < (0.8 + r13)*(2 + d13);
x14 < (0.8 + r14)*(4 + d14);
x24 < (0.8 + r24)*(5 + d24);
x3t < (0.8 + r3t)*(4 + d3t);
x4t < (0.8 + r4t)*(7 + d4t);

! capacity improvement budget;
5*ds1 + 6*ds2 + 2*d13 + 4*d14 + 5*d24 + 4*d3t + 7*d4t <= 20;

.8*rs1 + .8*r13 + .8*r3t + .8*r14 + .8*rs2 + .8*r24 + .8*r4t <= 0.24;

rs1 < 0.2;
rs2 < 0.2;
r13 < 0.2;
r14 < 0.2;
r24 < 0.2;
r3t < 0.2;
r4t < 0.2;

END
```

OUTPUT FOR SAMPLE NETWORK: MODEL A

Rows= 23 Vars= 22 No. integer vars= 0
Nonlinear rows= 7 Nonlinear vars= 14 Nonlinear constraints= 7
Nonzeros= 75 Constraint nonz= 58 Density=0.142

Optimal solution found at step: 9
Objective value: 10.70979

Variable	Value	Reduced Cost
V	10.70979	0.0000000E+00
XS1	6.709787	0.0000000E+00
XS2	4.000000	0.0000000E+00
X13	3.509787	0.0000000E+00
X14	3.200000	0.0000000E+00
X24	4.000000	0.0000000E+00
X3T	3.509787	0.0000000E+00
X4T	7.200000	0.0000000E+00
RS1	0.1000000	0.0000000E+00
DS1	2.455319	0.0000000E+00
RS2	0.0000000E+00	3.172476
DS2	0.0000000E+00	0.4595745
R13	0.0000000E+00	2.332368
D13	2.387234	0.0000000E+00
R14	0.0000000E+00	3.019285
D14	0.0000000E+00	0.2757447
R24	0.0000000E+00	0.8533273
D24	0.0000000E+00	0.1191492E-01
R3T	0.0000000E+00	1.492259
D3T	0.3872340	0.0000000E+00
R4T	0.2000000	0.0000000E+00
D4T	0.2000000	0.0000000E+00

Row	Slack or Surplus	Dual Price
1	10.70979	1.000000
2	0.0000000E+00	0.0000000E+00
3	0.0000000E+00	0.4255319
4	0.0000000E+00	0.0000000E+00
5	0.0000000E+00	0.6170213
6	0.0000000E+00	0.4638298
7	0.0000000E+00	1.000000
8	0.0000000E+00	0.4255319
9	0.8000000	0.0000000E+00
10	0.0000000E+00	0.1914894
11	0.0000000E+00	0.3829787E-01
12	0.0000000E+00	0.4638298
13	0.0000000E+00	0.3829787
14	0.0000000E+00	0.5361702
15	0.0000000E+00	0.7659575E-01
16	0.0000000E+00	3.965595

17	0.1000000	0.0000000E+00
18	0.2000000	0.0000000E+00
19	0.2000000	0.0000000E+00
20	0.2000000	0.0000000E+00
21	0.2000000	0.0000000E+00
22	0.2000000	0.0000000E+00
23	0.0000000E+00	0.6879492

```
! Model B maximizes flow damage utility without the hardening game;

MODEL:

max = Vef;

! value function for flow damage utility;
Vef < V - xs1;
Vef < V - xs2;
Vef < V - x13;
Vef < V - x14;
Vef < V - x24;
Vef < V - x3t;
Vef < V - x4t;

xs1 + xs2 - V = 0;
x13 + x14 - xs1 = 0;
x24 - xs2 = 0;
x3t - x13 = 0;
x4t - x14 - x24 = 0;
V - x3t - x4t = 0;

xs1 < (0.8 + rs1)*(5 + ds1);
xs2 < (0.8 + rs2)*(6 + ds2);
x13 < (0.8 + r13)*(2 + d13);
x14 < (0.8 + r14)*(4 + d14);
x24 < (0.8 + r24)*(5 + d24);
x3t < (0.8 + r3t)*(4 + d3t);
x4t < (0.8 + r4t)*(7 + d4t);

5*ds1 + 6*ds2 + 2*d13 + 4*d14 + 5*d24 + 4*d3t + 7*d4t <= 20;

.8*rs1 + .8*r13 + .8*r3t + .8*r14 + .8*rs2 + .8*r24 + .8*r4t <= 0.24;

rs1 < 0.2;
rs2 < 0.2;
r13 < 0.2;
r14 < 0.2;
r24 < 0.2;
r3t < 0.2;
r4t < 0.2;

END
```

OUTPUT FOR SAMPLE NETWORK: MODEL B

Rows= 30 Vars= 23 No. integer vars= 0
Nonlinear rows= 7 Nonlinear vars= 14 Nonlinear constraints= 7
Nonzeros= 96 Constraint nonz= 79 Density=0.133

Optimal solution found at step: 12
Objective value: 4.954854

Variable	Value	Reduced Cost
VEF	4.954854	0.0000000E+00
V	9.909708	0.0000000E+00
XS1	4.954854	0.0000000E+00
XS2	4.954854	0.0000000E+00
X13	4.954854	0.0000000E+00
X14	0.0000000E+00	0.0000000E+00
X24	4.954854	0.0000000E+00
X3T	4.954854	0.0000000E+00
X4T	4.954854	0.0000000E+00
RS1	0.1239241	0.0000000E+00
DS1	0.3628369	-0.3167755E-05
RS2	0.2580902E-01	0.0000000E+00
DS2	0.0000000E+00	0.8719221E-01
R13	0.0000000E+00	0.5886117
D13	4.193568	0.0000000E+00
R14	0.0000000E+00	1.261801
D14	0.0000000E+00	0.1739066
R24	0.1239173	0.0000000E+00
D24	0.3628759	0.0000000E+00
R3T	0.2634957E-01	0.0000000E+00
D3T	1.996075	0.1146281E-04
R4T	0.0000000E+00	1.261801
D4T	0.0000000E+00	0.3043366

Row	Slack or Surplus	Dual Price
1	4.954854	1.000000
2	0.0000000E+00	0.4455845
3	0.0000000E+00	0.2352861
4	0.0000000E+00	0.0000000E+00
5	4.954854	0.0000000E+00
6	0.0000000E+00	0.0000000E+00
7	0.0000000E+00	0.0000000E+00
8	0.0000000E+00	0.3191294
9	0.0000000E+00	0.0000000E+00
10	0.0000000E+00	0.6808705
11	0.0000000E+00	0.4455862
12	0.0000000E+00	0.7895622
13	0.0000000E+00	0.6808705
14	0.0000000E+00	1.000000
15	0.0000000E+00	0.2352861

16	0.0000000E+00	0.2103001
17	0.0000000E+00	0.1086916
18	3.200000	0.0000000E+00
19	0.0000000E+00	0.2352844
20	0.0000000E+00	0.2104378
21	0.6451459	0.0000000E+00
22	0.0000000E+00	0.4347666E-01
23	0.0000000E+00	1.577251
24	0.7607593E-01	0.0000000E+00
25	0.1741910	0.0000000E+00
26	0.2000000	0.0000000E+00
27	0.2000000	0.0000000E+00
28	0.7608266E-01	0.0000000E+00
29	0.1736504	0.0000000E+00
30	0.2000000	0.0000000E+00

```
! Model C maximizes flow damage utility with the hardening game;

MODEL:

MAX = Vef;

! flow damage utility value function with game variables;
Vef < V - xs1*(1 - ys1);
Vef < V - xs2*(1 - ys2);
Vef < V - x13*(1 - y13);
Vef < V - x14*(1 - y14);
Vef < V - x24*(1 - y24);
Vef < V - x3t*(1 - y3t);
Vef < V - x4t*(1 - y4t);

xs1 + xs2 - V = 0;
x13 + x14 - xs1 = 0;
x24 - xs2 = 0;
x3t - x13 = 0;
x4t - x14 - x24 = 0;
V - x3t - x4t = 0;

xs1 < (0.8 + rs1)*(5 + ds1);
xs2 < (0.8 + rs2)*(6 + ds2);
x13 < (0.8 + r13)*(2 + d13);
x14 < (0.8 + r14)*(4 + d14);
x24 < (0.8 + r24)*(5 + d24);
x3t < (0.8 + r3t)*(4 + d3t);
x4t < (0.8 + r4t)*(7 + d4t);

5*ds1 + 6*ds2 + 2*d13 + 4*d14 + 5*d24 + 4*d3t + 7*d4t <= 20;

.8*rs1 + .8*r13 + .8*r3t + .8*r14 + .8*rs2 + .8*r24 + .8*r4t <= 0.24;

! game budget;
ys1 + ys2 + y13 + y14 + y24 + y3t + y4t <= 1;

rs1 < 0.2;
rs2 < 0.2;
r13 < 0.2;
r14 < 0.2;
r24 < 0.2;
r3t < 0.2;
r4t < 0.2;

END
```

OUTPUT FOR SAMPLE NETWORK: MODEL C

Rows= 31 Vars= 30 No. integer vars= 0
Nonlinear rows= 14 Nonlinear vars= 28 Nonlinear constraints= 14
Nonzeros= 111 Constraint nonz= 93 Density=0.116

Optimal solution found at step: 16
Objective value: 6.946994

Variable	Value	Reduced Cost
VEF	6.946994	0.0000000E+00
V	10.58830	0.0000000E+00
XS1	6.841311	0.0000000E+00
YS1	0.4677466	0.0000000E+00
XS2	3.746994	0.0000000E+00
YS2	0.2820484E-01	0.0000000E+00
X13	3.641311	0.0000000E+00
Y13	0.0000000E+00	1.070211
X14	3.200000	0.0000000E+00
Y14	0.0000000E+00	1.070211
X24	3.746994	0.0000000E+00
Y24	0.2820484E-01	0.0000000E+00
X3T	3.641311	0.0000000E+00
Y3T	0.0000000E+00	0.6395315
X4T	6.946994	0.0000000E+00
Y4T	0.4758437	0.0000000E+00
RS1	0.1075723	0.0000000E+00
DS1	2.538034	0.0000000E+00
RS2	0.0000000E+00	2.548886
DS2	0.0000000E+00	0.3682602
R13	0.0000000E+00	1.850474
D13	2.551638	0.0000000E+00
R14	0.0000000E+00	2.013981
D14	0.0000000E+00	0.1385258
R24	0.0000000E+00	2.548886
D24	0.0000000E+00	0.3068835
R3T	0.0000000E+00	1.152063
D3T	0.5516383	0.0000000E+00
R4T	0.1924277	0.0000000E+00
D4T	0.0000000E+00	0.6826763E-01

Row	Slack or Surplus	Dual Price
1	6.946994	1.000000
2	0.0000000E+00	0.1564336
3	0.0000000E+00	0.2856184
4	0.0000000E+00	0.0000000E+00
5	0.4413107	0.0000000E+00
6	0.0000000E+00	0.2856184
7	0.0000000E+00	0.1182758
8	0.0000000E+00	0.1540538

9	0.0000000E+00	0.0000000E+00
10	0.0000000E+00	0.4213990
11	0.0000000E+00	0.2775626
12	0.0000000E+00	0.5748407
13	0.0000000E+00	0.5551252
14	0.0000000E+00	1.000000
15	0.0000000E+00	0.3381367
16	1.053006	0.0000000E+00
17	0.0000000E+00	0.1534417
18	0.0000000E+00	0.1337263
19	0.2530060	0.0000000E+00
20	0.0000000E+00	0.3068835
21	0.0000000E+00	0.3641265
22	0.0000000E+00	0.6137669E-01
23	0.0000000E+00	3.186107
24	0.0000000E+00	1.070211
25	0.9242772E-01	0.0000000E+00
26	0.2000000	0.0000000E+00
27	0.2000000	0.0000000E+00
28	0.2000000	0.0000000E+00
29	0.2000000	0.0000000E+00
30	0.2000000	0.0000000E+00
31	0.7572283E-02	0.0000000E+00

```
! Model D is the MCO model for flow and reliability damage utilities
with the hardening game;

MODEL:

max = Vef;
Vrel >= 0.9;


Rst = (.8 + rs1)*(.8 + r13)*(.8 + r3t)
+ (.8 + rs1)*(.8 + r14)*(.8 + r4t)
+ (.8 + rs2)*(.8 + r24)*(.8 + r4t)
- (.8 + rs1)*(.8 + r13)*(.8 + r3t)*(.8 + r14)*(.8 + r4t)
- (.8 + rs1)*(.8 + r14)*(.8 + rs2)*(.8 + r24)*(.8 + r4t)
- (.8 + rs1)*(.8 + r13)*(.8 + r3t)*(.8 + rs2)*(.8 + r24)*(.8 + r4t)
+ (.8 + rs1)*(.8 + r13)*(.8 + r3t)*(.8 + r14)*(.8 + rs2)*(.8 + r24)*(.8
+ r4t);

RA = (.8 + rs2)*(.8 + r24)*(.8 + r4t);
RB = (.8 + rs1)*(.8 + r14)*(.8 + r4t)
+ (.8 + rs2)*(.8 + r24)*(.8 + r4t)
- (.8 + rs1)*(.8 + r14)*(.8 + rs2)*(.8 + r24)*(.8 + r4t);
RC = RB;
RD = (.8 + rs1)*(.8 + r13)*(.8 + r3t)
+ (.8 + rs2)*(.8 + r24)*(.8 + r4t)
- (.8 + rs1)*(.8 + r13)*(.8 + r3t)*(.8 + rs2)*(.8 + r24)*(.8 + r4t);
RE = (.8 + rs1)*(.8 + r13)*(.8 + r3t)
+ (.8 + rs1)*(.8 + r14)*(.8 + r4t)
- (.8 + rs1)*(.8 + r13)*(.8 + r3t)*(.8 + r14)*(.8 + r4t);
RF = RE;
RG = (.8 + rs1)*(.8 + r13)*(.8 + r3t);

Vrel < (RA - Rst + 1)*(1 - ys1) + ys1;
Vrel < (RB - Rst + 1)*(1 - y13) + y13;
Vrel < (RC - Rst + 1)*(1 - y3t) + y3t;
Vrel < (RD - Rst + 1)*(1 - y14) + y14;
Vrel < (RE - Rst + 1)*(1 - ys2) + ys2;
Vrel < (RF - Rst + 1)*(1 - y24) + y24;
Vrel < (RG - Rst + 1)*(1 - y4t) + y4t;

ys1 + ys2 + y13 + y14 + y24 + y3t + y4t <= 1;

5*ds1 + 6*ds2 + 2*d13 + 4*d14 + 5*d24 + 4*d3t + 7*d4t < Bcap;

.8*rs1 + .8*r13 + .8*r3t + .8*r14 + .8*rs2 + .8*r24 + .8*r4t < Brel;

0.0502*Bcap + Brel <= 0.87;

Vef < V - xs1*(1 - ys1);
Vef < V - xs2*(1 - ys2);
Vef < V - x13*(1 - y13);
Vef < V - x14*(1 - y14);
```

```
Vef < V - x24*(1 - y24);
Vef < V - x3t*(1 - y3t);
Vef < V - x4t*(1 - y4t);

xs1 + xs2 - V = 0;
x13 + x14 - xs1 = 0;
x24 - xs2 = 0;
x3t - x13 = 0;
x4t - x14 - x24 = 0;
V - x3t - x4t = 0;

xs1 < (0.8 + rs1)*(5 + ds1);
xs2 < (0.8 + rs2)*(6 + ds2);
x13 < (0.8 + r13)*(2 + d13);
x14 < (0.8 + r14)*(4 + d14);
x24 < (0.8 + r24)*(5 + d24);
x3t < (0.8 + r3t)*(4 + d3t);
x4t < (0.8 + r4t)*(7 + d4t);

rs1 < 0.2;
rs2 < 0.2;
r13 < 0.2;
r14 < 0.2;
r24 < 0.2;
r3t < 0.2;
r4t < 0.2;

END
```

OUTPUT FOR SAMPLE NETWORK: MODEL D

Rows= 48 Vars= 41 No. integer vars= 0
Nonlinear rows= 27 Nonlinear vars= 36 Nonlinear constraints= 27
Nonzeros= 196 Constraint nonz= 165 Density=0.097

Optimal solution found at step: 28
Objective value: 6.086774

Variable	Value	Reduced Cost
VEF	6.086774	0.0000000E+00
VREL	0.9000000	0.0000000E+00
RST	0.9953634	0.0000000E+00
RS1	0.2000000	0.0000000E+00
R13	0.2000000	0.0000000E+00
R3T	0.2346866E-01	0.0000000E+00
R14	0.0000000E+00	1.937900
R4T	0.2000000	0.0000000E+00
RS2	0.6867618E-01	0.0000000E+00
R24	0.2000000	0.0000000E+00
RA	0.8686762	0.0000000E+00
RB	0.9737352	0.0000000E+00
RC	0.9737352	0.0000000E+00
RD	0.9768172	0.0000000E+00
RE	0.9646937	0.0000000E+00
RF	0.9646937	0.0000000E+00
RG	0.8234687	0.0000000E+00
YS1	0.2106546	0.0000000E+00
Y13	0.0000000E+00	1.553004
Y3T	0.0000000E+00	1.553004
Y14	0.0000000E+00	1.553004
YS2	0.1855483	0.0000000E+00
Y24	0.1855483	0.0000000E+00
Y4T	0.4182488	0.0000000E+00
DS1	0.1590328	0.0000000E+00
DS2	0.0000000E+00	0.6952680
D13	1.159033	0.0000000E+00
D14	0.0000000E+00	0.4635120
D24	0.0000000E+00	0.3170845
D3T	0.0000000E+00	0.4635120
D4T	0.0000000E+00	0.6605702
BCAP	3.113229	0.0000000E+00
BREL	0.7137159	0.0000000E+00
V	10.15903	0.0000000E+00
XS1	5.159033	0.0000000E+00
XS2	5.000000	0.0000000E+00
X13	3.159033	0.0000000E+00
X14	2.000000	0.0000000E+00
X24	5.000000	0.0000000E+00
X3T	3.159033	0.0000000E+00

X4T	7.000000	0.0000000E+00

Row	Slack or Surplus	Dual Price
1	6.086774	1.000000
2	0.0000000E+00	-5.867519
3	0.0000000E+00	-3.935644
4	0.0000000E+00	1.985614
5	0.0000000E+00	0.0000000E+00
6	0.0000000E+00	0.0000000E+00
7	0.0000000E+00	0.0000000E+00
8	0.0000000E+00	0.0000000E+00
9	0.0000000E+00	0.0000000E+00
10	0.0000000E+00	1.950030
11	0.0000000E+00	2.515520
12	0.7837179E-01	0.0000000E+00
13	0.7837179E-01	0.0000000E+00
14	0.8145378E-01	0.0000000E+00
15	0.7502100E-01	0.0000000E+00
16	0.7502100E-01	0.0000000E+00
17	0.0000000E+00	3.352000
18	0.0000000E+00	1.553004
19	0.0000000E+00	0.1158780
20	0.0000000E+00	2.308326
21	0.0000000E+00	2.308326
22	0.0000000E+00	0.2392540
23	0.0000000E+00	0.3106007
24	0.9132258	0.0000000E+00
25	2.072259	0.0000000E+00
26	0.0000000E+00	0.3106007
27	0.9132258	0.0000000E+00
28	0.0000000E+00	0.1395446
29	0.0000000E+00	-0.2529693
30	0.0000000E+00	0.5152748
31	0.0000000E+00	0.0000000E+00
32	0.0000000E+00	0.7470307
33	0.0000000E+00	0.5152748
34	0.0000000E+00	0.7470307
35	0.0000000E+00	0.5793900
36	0.2120571	0.0000000E+00
37	0.0000000E+00	0.2317560
38	1.200000	0.0000000E+00
39	0.0000000E+00	0.2623055
40	0.1348419	0.0000000E+00
41	0.0000000E+00	0.1505757
42	0.0000000E+00	2.249623
43	0.1313238	0.0000000E+00
44	0.0000000E+00	0.4061311
45	0.2000000	0.0000000E+00
46	0.0000000E+00	1.069017
47	0.1765313	0.0000000E+00
48	0.0000000E+00	0.2557080

SUMMARIZED OUTPUT FOR SAMPLE NETWORK: MODEL D

"Expensive" capacity.						
Vef	Vrel	range equalized Vef	range equalized Vrel	l-1	l-2	l-inf
5.61	0.781	0.000	1.000	1.000	1.000	1.000
5.60	0.800	0.008	0.913	0.921	0.913	0.913
5.58	0.820	0.023	0.822	0.845	0.822	0.822
5.57	0.840	0.031	0.731	0.761	0.731	0.731
5.56	0.860	0.038	0.639	0.677	0.640	0.639
5.55	0.880	0.046	0.548	0.594	0.550	0.548
5.53	0.900	0.061	0.457	0.518	0.461	0.457
5.51	0.920	0.076	0.365	0.442	0.373	0.365
5.50	0.926	0.084	0.338	0.422	0.348	0.338
5.49	0.930	0.092	0.320	0.411	0.333	0.320
5.49	0.933	0.092	0.306	0.398	0.319	0.306
5.48	0.941	0.099	0.269	0.369	0.287	0.269
5.47	0.95	0.107	0.228	0.335	0.252	0.228
5.46	0.956	0.115	0.201	0.315	0.231	0.201
5.45	0.964	0.122	0.164	0.287	0.205	0.164
5.44	0.97	0.130	0.137	0.267	0.189	0.137
5.4	0.972	0.160	0.128	0.288	0.205	0.160
5.2	0.977	0.313	0.105	0.418	0.330	0.313
5	0.984	0.466	0.073	0.539	0.471	0.466
4.5	0.993	0.847	0.032	0.879	0.848	0.847
4.3	1	1.000	0.000	1.000	1.000	1.000

alpha value = 0.502

SUMMARIZED OUTPUT FOR SAMPLE NETWORK: MODEL D (CONT.)

"Neutrality" between capacity and reliability improvement costs.						
Vef	*Vrel*	*range equalized Vef*	*range equalized Vrel*	*l-1*	*l-2*	*l-inf*
6.56	0.810	0.000	1.000	1.000	1.000	1.000
6.52	0.820	0.019	0.947	0.966	0.948	0.947
6.40	0.847	0.074	0.805	0.879	0.809	0.805
6.26	0.860	0.139	0.737	0.876	0.750	0.737
6.19	0.880	0.171	0.632	0.803	0.654	0.632
6.09	0.900	0.218	0.526	0.744	0.570	0.526
5.97	0.920	0.273	0.421	0.694	0.502	0.421
5.76	0.940	0.370	0.316	0.686	0.487	0.370
5.70	0.945	0.398	0.289	0.688	0.492	0.398
5.60	0.953	0.444	0.247	0.692	0.509	0.444
5.50	0.962	0.491	0.200	0.691	0.530	0.491
5.40	0.968	0.537	0.168	0.705	0.563	0.537
5.30	0.971	0.583	0.153	0.736	0.603	0.583
5.10	0.971	0.676	0.153	0.829	0.693	0.676
5.00	0.982	0.722	0.095	0.817	0.728	0.722
4.90	0.983	0.769	0.089	0.858	0.774	0.769
4.80	0.988	0.815	0.063	0.878	0.817	0.815
4.70	0.994	0.861	0.032	0.893	0.862	0.861
4.40	1.000	1.000	0.000	1.000	1.000	1.000

alpha value = 0.0502

SUMMARIZED OUTPUT FOR SAMPLE NETWORK: MODEL D (CONT.)

"Cheap" capacity.						
Vef	*Vrel*	*range equalized Vef*	*range equalized Vrel*	*l-1*	*l-2*	*l-inf*
12.34	0.840	0.000	1.000	1.000	1.000	1.000
11.89	0.850	0.065	0.938	1.002	0.940	0.938
11.34	0.860	0.144	0.875	1.019	0.887	0.875
10.79	0.870	0.223	0.813	1.036	0.843	0.813
10.31	0.880	0.293	0.750	1.043	0.805	0.750
9.85	0.890	0.359	0.688	1.046	0.775	0.688
9.25	0.900	0.445	0.625	1.070	0.767	0.625
8.89	0.910	0.497	0.563	1.060	0.751	0.563
8.45	0.920	0.561	0.500	1.061	0.751	0.561
8.44	0.930	0.562	0.438	0.999	0.712	0.562
7.71	0.940	0.667	0.375	1.042	0.765	0.667
7.20	0.954	0.741	0.288	1.028	0.794	0.741
7.00	0.960	0.769	0.250	1.019	0.809	0.769
6.90	0.967	0.784	0.206	0.990	0.811	0.784
6.70	0.968	0.813	0.200	1.013	0.837	0.813
6.60	0.970	0.827	0.188	1.015	0.848	0.827
6.50	0.973	0.841	0.169	1.010	0.858	0.841
6.40	0.976	0.856	0.150	1.006	0.869	0.856
6.30	0.977	0.870	0.144	1.014	0.882	0.870
6.20	0.981	0.885	0.119	1.003	0.893	0.885
6.00	0.982	0.914	0.113	1.026	0.920	0.914
5.90	0.990	0.928	0.063	0.990	0.930	0.928
5.40	1.000	1.000	0.000	1.000	1.000	1.000

alpha value = 0.00502

Appendix B: LINGO Input/Output Files for Networks A, B, and C

Appendix B contains input and output examples using Networks A, B, and C in the case of Model C, and Network A only in the case of Model D. In order:

- Model C, Network A
- Model C, Network B
- Model C, Network C
- Model D, Network A (including a table with the data for efficient frontier calculation)

```
MODEL:

! Model C for Network A;

! Since LINGO numbers from 1 rather than 0, subtract 1 from variables to
translate
back to diagram.  For example, arc(1,2) in LINGO program = arc (s,1)
actual.;

SETS:
      NODE / 1..17 / : rel_node, imp_rel_node;
      ARC(NODE, NODE) / 1,2 1,3 1,4 1,5 2,13 2,14 2,16 3,6 3,15 4,10
4,12 5,15
            6,11 6,12 7,15 8,15 9,15 10,16 11,16 12,16 13,16 14,16 15,17
            16,7 16,8 16,9 / :
      cap, rel_arc, imp_cap, imp_rel_arc, flow, y_arc;
ENDSETS

DATA:

      rel_node = 1 1 .3 .7 .5 .8 1 .3 .7 .5 .8 1 .3 .7 .5 .8 1;
      rel_arc = 1 1 1 1 1 1 .7 .3 .6 1 1 1 .6 .7 .6 .6 .3 1 .6 1 .7 1 1
.3 .6 .7;
      cap = 0 0 0 0
            1200 1200 1200 1200 1200 1200 1200 1200 1200 1200 4800 4800
4800 4800 4800
            4800 4800 4800 0 4800 4800 4800;
ENDDATA

MAX = Vef;

@FOR(NODE(I) : rel_node(I) + imp_rel_node(I) <= 1);

@FOR(ARC(I,J) : rel_arc(I,J) + imp_rel_arc(I,J) <= 1);

@FOR(ARC(I,J) | I #NE# 1 #AND# J #NE# 17 :
      flow(I,J) - ((cap(I,J) + imp_cap(I,J))*(rel_arc(I,J) +
imp_rel_arc(I,J))
      *(rel_node(J) + imp_rel_node(J))) <= 0);


@FOR(ARC(I,J) | I #NE# 1 #AND# J #NE# 17 :
      Vef - V + flow(I,J)*(1 - y_arc(I,J)) <= 0);


@SUM(ARC(I,J) | I #NE# 1 #AND# J #NE# 17 :
      cap(I,J)*imp_cap(I,J)) <= Bcap;

@SUM(ARC(I,J) : rel_arc(I,J)*imp_rel_arc(I,J))
      + @SUM(NODE(K):rel_node(K)*imp_rel_node(K)) <= Brel;

Bcap = 1000000;
```

```
Brel = .05;

@SUM(ARC(I,J) : y_arc(I,J)) <= 1;

@FOR(NODE(J) | J #EQ# 1 :
      @SUM(ARC(J,K):flow(J,K)) - V = 0);

@FOR(NODE(J) | J #GT# 1 #AND# J #LT# 17 :
      @SUM(ARC(J,K):flow(J,K)) - @SUM(ARC(I,J):flow(I,J)) = 0);

@FOR(NODE(J) | J #EQ# 17 :
      V - @SUM(ARC(I,J):flow(I,J)) = 0);

END
```

OUTPUT FOR NETWORK A: MODEL C

Rows= 106 Vars= 123 No. integer vars= 0
Nonlinear rows= 42 Nonlinear vars= 95 Nonlinear constraints= 42
Nonzeros= 405 Constraint nonz= 355 Density=0.031

Optimal solution found at step: 40
Objective value: 3974.580

Variable	Value	Reduced Cost
VEF	3974.580	0.0000000E+00
V	4890.115	0.0000000E+00
BCAP	1000000.	0.0000000E+00
BREL	0.5000000E-01	0.0000000E+00
REL_NODE(1)	1.000000	0.0000000E+00
REL_NODE(2)	1.000000	0.0000000E+00
REL_NODE(3)	0.3000000	0.0000000E+00
REL_NODE(4)	0.7000000	0.0000000E+00
REL_NODE(5)	0.5000000	0.0000000E+00
REL_NODE(6)	0.8000000	0.0000000E+00
REL_NODE(7)	1.000000	0.0000000E+00
REL_NODE(8)	0.3000000	0.0000000E+00
REL_NODE(9)	0.7000000	0.0000000E+00
REL_NODE(10)	0.5000000	0.0000000E+00
REL_NODE(11)	0.8000000	0.0000000E+00
REL_NODE(12)	1.000000	0.0000000E+00
REL_NODE(13)	0.3000000	0.0000000E+00
REL_NODE(14)	0.7000000	0.0000000E+00
REL_NODE(15)	0.5000000	0.0000000E+00
REL_NODE(16)	0.8000000	0.0000000E+00
REL_NODE(17)	1.000000	0.0000000E+00
IMP_REL_NODE(1)	0.0000000E+00	8554.369
IMP_REL_NODE(2)	0.0000000E+00	8554.369
IMP_REL_NODE(3)	0.0000000E+00	2566.311
IMP_REL_NODE(4)	0.0000000E+00	5988.058
IMP_REL_NODE(5)	0.0000000E+00	4277.185
IMP_REL_NODE(6)	0.2499936E-14	6843.495
IMP_REL_NODE(7)	0.0000000E+00	7718.226
IMP_REL_NODE(8)	0.1789413E-01	0.0000000E+00
IMP_REL_NODE(9)	0.2499936E-14	5988.058
IMP_REL_NODE(10)	0.2499936E-14	4277.185
IMP_REL_NODE(11)	0.2499936E-14	6843.495
IMP_REL_NODE(12)	0.2499936E-14	8554.369
IMP_REL_NODE(13)	0.2499936E-14	2566.311
IMP_REL_NODE(14)	0.2499936E-14	5988.058
IMP_REL_NODE(15)	0.4302571E-01	0.0000000E+00
IMP_REL_NODE(16)	0.2499936E-14	6843.495
IMP_REL_NODE(17)	0.0000000E+00	8554.369
CAP(1, 2)	0.0000000E+00	0.0000000E+00
CAP(1, 3)	0.0000000E+00	0.0000000E+00

CAP(1, 4)	0.0000000E+00	0.0000000E+00
CAP(1, 5)	0.0000000E+00	0.0000000E+00
CAP(2, 13)	1200.000	0.0000000E+00
CAP(2, 14)	1200.000	0.0000000E+00
CAP(2, 16)	1200.000	0.0000000E+00
CAP(3, 6)	1200.000	0.0000000E+00
CAP(3, 15)	1200.000	0.0000000E+00
CAP(4, 10)	1200.000	0.0000000E+00
CAP(4, 12)	1200.000	0.0000000E+00
CAP(5, 15)	1200.000	0.0000000E+00
CAP(6, 11)	1200.000	0.0000000E+00
CAP(6, 12)	1200.000	0.0000000E+00
CAP(7, 15)	4800.000	0.0000000E+00
CAP(8, 15)	4800.000	0.0000000E+00
CAP(9, 15)	4800.000	0.0000000E+00
CAP(10, 16)	4800.000	0.0000000E+00
CAP(11, 16)	4800.000	0.0000000E+00
CAP(12, 16)	4800.000	0.0000000E+00
CAP(13, 16)	4800.000	0.0000000E+00
CAP(14, 16)	4800.000	0.0000000E+00
CAP(15, 17)	0.0000000E+00	0.0000000E+00
CAP(16, 7)	4800.000	0.0000000E+00
CAP(16, 8)	4800.000	0.0000000E+00
CAP(16, 9)	4800.000	0.0000000E+00
REL_ARC(1, 2)	1.000000	0.0000000E+00
REL_ARC(1, 3)	1.000000	0.0000000E+00
REL_ARC(1, 4)	1.000000	0.0000000E+00
REL_ARC(1, 5)	1.000000	0.0000000E+00
REL_ARC(2, 13)	1.000000	0.0000000E+00
REL_ARC(2, 14)	1.000000	0.0000000E+00
REL_ARC(2, 16)	0.7000000	0.0000000E+00
REL_ARC(3, 6)	0.3000000	0.0000000E+00
REL_ARC(3, 15)	0.6000000	0.0000000E+00
REL_ARC(4, 10)	1.000000	0.0000000E+00
REL_ARC(4, 12)	1.000000	0.0000000E+00
REL_ARC(5, 15)	1.000000	0.0000000E+00
REL_ARC(6, 11)	0.6000000	0.0000000E+00
REL_ARC(6, 12)	0.7000000	0.0000000E+00
REL_ARC(7, 15)	0.6000000	0.0000000E+00
REL_ARC(8, 15)	0.6000000	0.0000000E+00
REL_ARC(9, 15)	0.3000000	0.0000000E+00
REL_ARC(10, 16)	1.000000	0.0000000E+00
REL_ARC(11, 16)	0.6000000	0.0000000E+00
REL_ARC(12, 16)	1.000000	0.0000000E+00
REL_ARC(13, 16)	0.7000000	0.0000000E+00
REL_ARC(14, 16)	1.000000	0.0000000E+00
REL_ARC(15, 17)	1.000000	0.0000000E+00
REL_ARC(16, 7)	0.3000000	0.0000000E+00
REL_ARC(16, 8)	0.6000000	0.0000000E+00
REL_ARC(16, 9)	0.7000000	0.0000000E+00
IMP_CAP(1, 2)	0.0000000E+00	0.0000000E+00
IMP_CAP(1, 3)	0.0000000E+00	0.0000000E+00

IMP_CAP(1, 4)	0.0000000E+00	0.0000000E+00
IMP_CAP(1, 5)	0.0000000E+00	0.0000000E+00
IMP_CAP(2, 13)	0.0000000E+00	0.3795647
IMP_CAP(2, 14)	0.2499936E-14	0.3795647
IMP_CAP(2, 16)	0.2499936E-14	0.3795647
IMP_CAP(3, 6)	0.2499936E-14	0.3795647
IMP_CAP(3, 15)	0.2499936E-14	0.5374960E-01
IMP_CAP(4, 10)	0.2499936E-14	0.3795647
IMP_CAP(4, 12)	0.2499936E-14	0.3795647
IMP_CAP(5, 15)	833.3333	0.0000000E+00
IMP_CAP(6, 11)	0.0000000E+00	0.3795647
IMP_CAP(6, 12)	0.2499936E-14	0.3795647
IMP_CAP(7, 15)	0.2499936E-14	1.464413
IMP_CAP(8, 15)	0.2499936E-14	1.518259
IMP_CAP(9, 15)	0.0000000E+00	1.330480
IMP_CAP(10, 16)	0.2499936E-14	1.518259
IMP_CAP(11, 16)	0.0000000E+00	1.518259
IMP_CAP(12, 16)	0.0000000E+00	1.518259
IMP_CAP(13, 16)	0.0000000E+00	1.518259
IMP_CAP(14, 16)	0.4999871E-14	1.518259
IMP_CAP(15, 17)	0.0000000E+00	0.0000000E+00
IMP_CAP(16, 7)	0.2499936E-14	1.344056
IMP_CAP(16, 8)	0.0000000E+00	1.348291
IMP_CAP(16, 9)	0.1999946E-13	1.518259
IMP_REL_ARC(1, 2)	0.0000000E+00	8554.369
IMP_REL_ARC(1, 3)	0.0000000E+00	8554.369
IMP_REL_ARC(1, 4)	0.0000000E+00	8554.369
IMP_REL_ARC(1, 5)	0.0000000E+00	8554.369
IMP_REL_ARC(2, 13)	0.0000000E+00	8554.369
IMP_REL_ARC(2, 14)	0.0000000E+00	8554.369
IMP_REL_ARC(2, 16)	0.2499936E-14	5988.058
IMP_REL_ARC(3, 6)	0.0000000E+00	2566.311
IMP_REL_ARC(3, 15)	0.2499936E-14	4480.991
IMP_REL_ARC(4, 10)	0.0000000E+00	8554.369
IMP_REL_ARC(4, 12)	0.2499936E-14	8554.369
IMP_REL_ARC(5, 15)	0.0000000E+00	7782.588
IMP_REL_ARC(6, 11)	0.2499936E-14	5132.621
IMP_REL_ARC(6, 12)	0.2499936E-14	5988.058
IMP_REL_ARC(7, 15)	0.2499936E-14	4701.870
IMP_REL_ARC(8, 15)	0.2499936E-14	5132.621
IMP_REL_ARC(9, 15)	0.5124759E-01	0.0000000E+00
IMP_REL_ARC(10, 16)	0.2499936E-14	8554.369
IMP_REL_ARC(11, 16)	0.2499936E-14	5132.621
IMP_REL_ARC(12, 16)	0.2499936E-14	8554.369
IMP_REL_ARC(13, 16)	0.2499936E-14	5988.058
IMP_REL_ARC(14, 16)	0.2499936E-14	8554.369
IMP_REL_ARC(15, 17)	0.0000000E+00	8554.369
IMP_REL_ARC(16, 7)	0.2581543E-01	0.0000000E+00
IMP_REL_ARC(16, 8)	0.2499936E-14	3772.930
IMP_REL_ARC(16, 9)	0.2499936E-14	5988.058
FLOW(1, 2)	1872.000	0.0000000E+00
FLOW(1, 3)	678.9785	0.0000000E+00

FLOW(1, 4)	1234.984	0.0000000E+00
FLOW(1, 5)	1104.152	0.0000000E+00
FLOW(2, 13)	360.0000	0.0000000E+00
FLOW(2, 14)	840.0000	0.0000000E+00
FLOW(2, 16)	672.0000	0.0000000E+00
FLOW(3, 6)	288.0000	0.0000000E+00
FLOW(3, 15)	390.9785	0.0000000E+00
FLOW(4, 10)	319.4491	0.0000000E+00
FLOW(4, 12)	915.5351	0.0000000E+00
FLOW(5, 15)	1104.152	0.0000000E+00
FLOW(6, 11)	288.0000	0.0000000E+00
FLOW(6, 12)	0.0000000E+00	0.0000000E+00
FLOW(7, 15)	1563.914	0.0000000E+00
FLOW(8, 15)	915.5351	0.0000000E+00
FLOW(9, 15)	915.5351	0.0000000E+00
FLOW(10, 16)	319.4491	0.0000000E+00
FLOW(11, 16)	288.0000	0.0000000E+00
FLOW(12, 16)	915.5351	0.0000000E+00
FLOW(13, 16)	360.0000	0.0000000E+00
FLOW(14, 16)	840.0000	0.0000000E+00
FLOW(15, 17)	4890.115	0.0000000E+00
FLOW(16, 7)	1563.914	0.0000000E+00
FLOW(16, 8)	915.5351	0.0000000E+00
FLOW(16, 9)	915.5351	0.0000000E+00
Y_ARC(1, 2)	0.0000000E+00	400.8452
Y_ARC(1, 3)	0.0000000E+00	400.8452
Y_ARC(1, 4)	0.0000000E+00	400.8452
Y_ARC(1, 5)	0.0000000E+00	400.8452
Y_ARC(2, 13)	0.0000000E+00	400.8452
Y_ARC(2, 14)	0.2499936E-14	400.8452
Y_ARC(2, 16)	0.2499936E-14	400.8452
Y_ARC(3, 6)	0.2499936E-14	400.8452
Y_ARC(3, 15)	0.2499936E-14	400.8452
Y_ARC(4, 10)	0.7499806E-14	400.8452
Y_ARC(4, 12)	0.0000000E+00	400.8452
Y_ARC(5, 15)	0.1708253	0.0000000E+00
Y_ARC(6, 11)	0.0000000E+00	400.8452
Y_ARC(6, 12)	0.2499936E-14	400.8452
Y_ARC(7, 15)	0.4145873	0.0000000E+00
Y_ARC(8, 15)	0.2499936E-14	301.1253
Y_ARC(9, 15)	0.0000000E+00	400.8452
Y_ARC(10, 16)	0.9355662E-14	400.8452
Y_ARC(11, 16)	0.2499936E-14	400.8452
Y_ARC(12, 16)	0.0000000E+00	400.8452
Y_ARC(13, 16)	0.2499936E-14	400.8452
Y_ARC(14, 16)	0.2299072E-13	400.8452
Y_ARC(15, 17)	0.0000000E+00	400.8452
Y_ARC(16, 7)	0.4145873	0.0000000E+00
Y_ARC(16, 8)	0.2499936E-14	400.8452
Y_ARC(16, 9)	0.0000000E+00	386.7206

Row	Slack or Surplus	Dual Price

1	3974.580	1.000000
2	0.0000000E+00	0.0000000E+00
3	0.0000000E+00	0.0000000E+00
4	0.7000000	0.0000000E+00
5	0.3000000	0.0000000E+00
6	0.5000000	0.0000000E+00
7	0.2000000	0.0000000E+00
8	0.0000000E+00	0.0000000E+00
9	0.6821059	0.0000000E+00
10	0.3000000	0.0000000E+00
11	0.5000000	0.0000000E+00
12	0.2000000	0.0000000E+00
13	0.0000000E+00	0.0000000E+00
14	0.7000000	0.0000000E+00
15	0.3000000	0.0000000E+00
16	0.4569743	0.0000000E+00
17	0.2000000	0.0000000E+00
18	0.0000000E+00	0.0000000E+00
19	0.0000000E+00	0.0000000E+00
20	0.0000000E+00	0.0000000E+00
21	0.0000000E+00	0.0000000E+00
22	0.0000000E+00	0.0000000E+00
23	0.0000000E+00	0.0000000E+00
24	0.0000000E+00	0.0000000E+00
25	0.3000000	0.0000000E+00
26	0.7000000	0.0000000E+00
27	0.4000000	0.0000000E+00
28	0.0000000E+00	0.0000000E+00
29	0.0000000E+00	0.0000000E+00
30	0.0000000E+00	0.0000000E+00
31	0.4000000	0.0000000E+00
32	0.3000000	0.0000000E+00
33	0.4000000	0.0000000E+00
34	0.4000000	0.0000000E+00
35	0.6487524	0.0000000E+00
36	0.0000000E+00	0.0000000E+00
37	0.4000000	0.0000000E+00
38	0.0000000E+00	0.0000000E+00
39	0.3000000	0.0000000E+00
40	0.0000000E+00	0.0000000E+00
41	0.0000000E+00	0.0000000E+00
42	0.6741846	0.0000000E+00
43	0.4000000	0.0000000E+00
44	0.3000000	0.0000000E+00
45	0.0000000E+00	0.0000000E+00
46	0.0000000E+00	0.0000000E+00
47	0.0000000E+00	0.0000000E+00
48	0.0000000E+00	0.0000000E+00
49	0.0000000E+00	1.000000
50	280.5509	0.0000000E+00
51	284.4649	0.0000000E+00
52	0.0000000E+00	0.6989811

53	288.0000	0.0000000E+00
54	840.0000	0.0000000E+00
55	0.0000000E+00	0.1652589
56	648.3790	0.0000000E+00
57	-0.2415829E-04	0.9845723
58	3520.551	0.0000000E+00
59	2016.000	0.0000000E+00
60	2924.465	0.0000000E+00
61	2328.000	0.0000000E+00
62	3000.000	0.0000000E+00
63	0.0000000E+00	0.5346481
64	0.0000000E+00	0.8910801
65	1436.465	0.0000000E+00
66	555.5351	0.0000000E+00
67	75.53510	0.0000000E+00
68	243.5351	0.0000000E+00
69	627.5351	0.0000000E+00
70	524.5566	0.0000000E+00
71	596.0860	0.0000000E+00
72	0.0000000E+00	0.0000000E+00
73	-0.1653406E-06	0.3630344
74	627.5351	0.0000000E+00
75	915.5351	0.0000000E+00
76	-0.4623219E-06	0.2563090
77	0.0000000E+00	0.1089199
78	0.0000000E+00	0.0000000E+00
79	596.0860	0.0000000E+00
80	627.5351	0.0000000E+00
81	0.0000000E+00	0.0000000E+00
82	555.5351	0.0000000E+00
83	75.53510	0.0000000E+00
84	0.1085472E-05	0.2563090
85	0.0000000E+00	0.0000000E+00
86	0.0000000E+00	0.1542770E-01
87	0.0000000E+00	0.3163039E-03
88	0.0000000E+00	8554.369
89	0.0000000E+00	0.3163039E-03
90	0.0000000E+00	8554.369
91	0.0000000E+00	400.8452
92	0.0000000E+00	0.0000000E+00
93	0.0000000E+00	0.0000000E+00
94	0.0000000E+00	0.0000000E+00
95	0.0000000E+00	0.0000000E+00
96	0.0000000E+00	0.0000000E+00
97	0.0000000E+00	0.0000000E+00
98	0.0000000E+00	0.6846946
99	0.0000000E+00	0.8910801
100	0.0000000E+00	0.1542770E-01
101	0.0000000E+00	0.0000000E+00
102	0.0000000E+00	0.0000000E+00
103	0.0000000E+00	0.0000000E+00
104	0.0000000E+00	0.0000000E+00

105	0.0000000E+00	0.0000000E+00
106	0.0000000E+00	1.000000
107	0.0000000E+00	0.0000000E+00
108	0.0000000E+00	1.000000

```
MODEL:

! Model C for Network B.

! Since LINGO numbers from 1 rather than 0, subtract 1 from variables to
translate back to diagram.  For example, arc(1,2) in LINGO program = arc
(s,1) actual.;


SETS:
      NODE / 1..28 / : rel_node, imp_rel_node;
      ARC(NODE, NODE) / 1,2 1,3 1,4 1,5 1,6 1,7 2,8 2,9 3,9 3,10 4,8
4,10 4,17
            5,17 5,25 6,15 7,17 7,26 7,27 8,11 9,11 10,11 11,12 11,13
11,14
            12,18 12,19 12,20 13,21 13,17 14,17 15,16 16,17 17,18 17,19
17,20
            17,21 17,22 17,23 17,24 17,25 17,26 17,27 18,28 19,28 20,28
21,28
            22,28 23,28 24,28 25,28 26,28 27,28 / :
      cap, rel_arc, imp_cap, imp_rel_arc, flow, y_arc;
ENDSETS

DATA:

      rel_node = 1 .7 .15 .03 1 1 .04 .04 1 .01 .7 .11 1 .06 .09 .18 .07
1 1 1 1
                  1 1 1 1 1 1 1;
      rel_arc = 1 1 1 1 1 1 .8 .8 .5 .5 .8 .5 .6 .8 .5 .8 .5 .6 .6 .5 .7
.5 .5 .7 .7
                  .5 .5 .5 .5 .7 .7 .8 .8 .6 .6 .6 .6 .6 .6 .6 .6 .6
1 1 1 1 1
                  1 1 1 1 1;
      cap = 0 0 0 0 0 0
            150 200 750 750 200 750 150 200 600 1200 1200 75 75 1200
1200 2400
            1200 1200 1200 1200 75 1200 1200 1200 600 1200 1200 75 75 75
75 75
            75 75 75 75 75
            0 0 0 0 0 0 0 0 0 0;
ENDDATA

MAX = Vef;


@FOR(NODE(I) : rel_node(I) + imp_rel_node(I) <= 1);

@FOR(ARC(I,J) : rel_arc(I,J) + imp_rel_arc(I,J) <= 1);

@FOR(ARC(I,J) | I #NE# 1 #AND# J #NE# 28 :
      flow(I,J) - ((cap(I,J) + imp_cap(I,J))*(rel_arc(I,J) +
imp_rel_arc(I,J))
```

```
      *(rel_node(J) + imp_rel_node(J))) <= 0);


@FOR(ARC(I,J) | I #NE# 1 #AND# J #NE# 28 :
      Vef - V + flow(I,J)*(1 - y_arc(I,J)) <= 0);

@SUM(ARC(I,J) | I #NE# 1 #AND# J #NE# 28 :
      cap(I,J)*imp_cap(I,J)) <= Bcap;

@SUM(ARC(I,J) : rel_arc(I,J)*imp_rel_arc(I,J))
      + @SUM(NODE(K):rel_node(K)*imp_rel_node(K)) <= Brel;

Brel <= 0.002;

Bcap <= 5000;

@SUM(ARC(I,J) : y_arc(I,J)) <= 1;

@FOR(NODE(J) | J #EQ# 1 :
      @SUM(ARC(J,K):flow(J,K)) - V = 0);

@FOR(NODE(J) | J #GT# 1 #AND# J #LT# 28 :
      @SUM(ARC(J,K):flow(J,K)) - @SUM(ARC(I,J):flow(I,J)) = 0);

@FOR(NODE(J) | J #EQ# 28 :
      V - @SUM(ARC(I,J):flow(I,J)) = 0);

END
```

OUTPUT FOR NETWORK B: MODEL C

Rows= 189 Vars= 244 No. integer vars= 0
Nonlinear rows= 74 Nonlinear vars= 168 Nonlinear constraints= 74
Nonzeros= 750 Constraint nonz= 660 Density=0.016

Optimal solution found at step: 53
Objective value: 885.3710

Variable	Value	Reduced Cost
VEF	885.3710	0.0000000E+00
V	1260.400	0.0000000E+00
BCAP	5000.000	0.0000000E+00
BREL	0.2000000E-02	0.0000000E+00
REL_NODE(1)	1.000000	0.0000000E+00
REL_NODE(2)	0.7000000	0.0000000E+00
REL_NODE(3)	0.1500000	0.0000000E+00
REL_NODE(4)	0.3000000E-01	0.0000000E+00
REL_NODE(5)	1.000000	0.0000000E+00
REL_NODE(6)	1.000000	0.0000000E+00
REL_NODE(7)	0.4000000E-01	0.0000000E+00
REL_NODE(8)	0.4000000E-01	0.0000000E+00
REL_NODE(9)	1.000000	0.0000000E+00
REL_NODE(10)	0.1000000E-01	0.0000000E+00
REL_NODE(11)	0.7000000	0.0000000E+00
REL_NODE(12)	0.1100000	0.0000000E+00
REL_NODE(13)	1.000000	0.0000000E+00
REL_NODE(14)	0.6000000E-01	0.0000000E+00
REL_NODE(15)	0.9000000E-01	0.0000000E+00
REL_NODE(16)	0.1800000	0.0000000E+00
REL_NODE(17)	0.7000000E-01	0.0000000E+00
REL_NODE(18)	1.000000	0.0000000E+00
REL_NODE(19)	1.000000	0.0000000E+00
REL_NODE(20)	1.000000	0.0000000E+00
REL_NODE(21)	1.000000	0.0000000E+00
REL_NODE(22)	1.000000	0.0000000E+00
REL_NODE(23)	1.000000	0.0000000E+00
REL_NODE(24)	1.000000	0.0000000E+00
REL_NODE(25)	1.000000	0.0000000E+00
REL_NODE(26)	1.000000	0.0000000E+00
REL_NODE(27)	1.000000	0.0000000E+00
REL_NODE(28)	1.000000	0.0000000E+00
IMP_REL_NODE(1)	0.0000000E+00	43279.08
IMP_REL_NODE(2)	0.0000000E+00	30295.36
IMP_REL_NODE(3)	0.0000000E+00	6491.862
IMP_REL_NODE(4)	0.0000000E+00	1298.372
IMP_REL_NODE(5)	0.0000000E+00	43279.08
IMP_REL_NODE(6)	0.0000000E+00	43279.08
IMP_REL_NODE(7)	0.0000000E+00	1731.163
IMP_REL_NODE(8)	0.0000000E+00	1569.588

IMP_REL_NODE(9)	0.0000000E+00	43101.80
IMP_REL_NODE(10)	0.2000000	0.0000000E+00
IMP_REL_NODE(11)	0.0000000E+00	30295.36
IMP_REL_NODE(12)	0.0000000E+00	4506.932
IMP_REL_NODE(13)	0.0000000E+00	43279.08
IMP_REL_NODE(14)	0.0000000E+00	2596.745
IMP_REL_NODE(15)	0.0000000E+00	3895.117
IMP_REL_NODE(16)	0.0000000E+00	7790.235
IMP_REL_NODE(17)	0.0000000E+00	846.2649
IMP_REL_NODE(18)	0.0000000E+00	43279.08
IMP_REL_NODE(19)	0.0000000E+00	43279.08
IMP_REL_NODE(20)	0.0000000E+00	43279.08
IMP_REL_NODE(21)	0.0000000E+00	43279.08
IMP_REL_NODE(22)	0.0000000E+00	43279.08
IMP_REL_NODE(23)	0.0000000E+00	43279.08
IMP_REL_NODE(24)	0.0000000E+00	43279.08
IMP_REL_NODE(25)	0.0000000E+00	42979.08
IMP_REL_NODE(26)	0.0000000E+00	43234.08
IMP_REL_NODE(27)	0.0000000E+00	43194.08
IMP_REL_NODE(28)	0.0000000E+00	43279.08
CAP(1, 2)	0.0000000E+00	0.0000000E+00
CAP(1, 3)	0.0000000E+00	0.0000000E+00
CAP(1, 4)	0.0000000E+00	0.0000000E+00
CAP(1, 5)	0.0000000E+00	0.0000000E+00
CAP(1, 6)	0.0000000E+00	0.0000000E+00
CAP(1, 7)	0.0000000E+00	0.0000000E+00
CAP(2, 8)	150.0000	0.0000000E+00
CAP(2, 9)	200.0000	0.0000000E+00
CAP(3, 9)	750.0000	0.0000000E+00
CAP(3, 10)	750.0000	0.0000000E+00
CAP(4, 8)	200.0000	0.0000000E+00
CAP(4, 10)	750.0000	0.0000000E+00
CAP(4, 17)	150.0000	0.0000000E+00
CAP(5, 17)	200.0000	0.0000000E+00
CAP(5, 25)	600.0000	0.0000000E+00
CAP(6, 15)	1200.000	0.0000000E+00
CAP(7, 17)	1200.000	0.0000000E+00
CAP(7, 26)	75.00000	0.0000000E+00
CAP(7, 27)	75.00000	0.0000000E+00
CAP(8, 11)	1200.000	0.0000000E+00
CAP(9, 11)	1200.000	0.0000000E+00
CAP(10, 11)	2400.000	0.0000000E+00
CAP(11, 12)	1200.000	0.0000000E+00
CAP(11, 13)	1200.000	0.0000000E+00
CAP(11, 14)	1200.000	0.0000000E+00
CAP(12, 18)	1200.000	0.0000000E+00
CAP(12, 19)	75.00000	0.0000000E+00
CAP(12, 20)	1200.000	0.0000000E+00
CAP(13, 17)	1200.000	0.0000000E+00
CAP(13, 21)	1200.000	0.0000000E+00
CAP(14, 17)	600.0000	0.0000000E+00
CAP(15, 16)	1200.000	0.0000000E+00

CAP(16, 17)	1200.000	0.0000000E+00
CAP(17, 18)	75.00000	0.0000000E+00
CAP(17, 19)	75.00000	0.0000000E+00
CAP(17, 20)	75.00000	0.0000000E+00
CAP(17, 21)	75.00000	0.0000000E+00
CAP(17, 22)	75.00000	0.0000000E+00
CAP(17, 23)	75.00000	0.0000000E+00
CAP(17, 24)	75.00000	0.0000000E+00
CAP(17, 25)	75.00000	0.0000000E+00
CAP(17, 26)	75.00000	0.0000000E+00
CAP(17, 27)	75.00000	0.0000000E+00
CAP(18, 28)	0.0000000E+00	0.0000000E+00
CAP(19, 28)	0.0000000E+00	0.0000000E+00
CAP(20, 28)	0.0000000E+00	0.0000000E+00
CAP(21, 28)	0.0000000E+00	0.0000000E+00
CAP(22, 28)	0.0000000E+00	0.0000000E+00
CAP(23, 28)	0.0000000E+00	0.0000000E+00
CAP(24, 28)	0.0000000E+00	0.0000000E+00
CAP(25, 28)	0.0000000E+00	0.0000000E+00
CAP(26, 28)	0.0000000E+00	0.0000000E+00
CAP(27, 28)	0.0000000E+00	0.0000000E+00
REL_ARC(1, 2)	1.000000	0.0000000E+00
REL_ARC(1, 3)	1.000000	0.0000000E+00
REL_ARC(1, 4)	1.000000	0.0000000E+00
REL_ARC(1, 5)	1.000000	0.0000000E+00
REL_ARC(1, 6)	1.000000	0.0000000E+00
REL_ARC(1, 7)	1.000000	0.0000000E+00
REL_ARC(2, 8)	0.8000000	0.0000000E+00
REL_ARC(2, 9)	0.8000000	0.0000000E+00
REL_ARC(3, 9)	0.5000000	0.0000000E+00
REL_ARC(3, 10)	0.5000000	0.0000000E+00
REL_ARC(4, 8)	0.8000000	0.0000000E+00
REL_ARC(4, 10)	0.5000000	0.0000000E+00
REL_ARC(4, 17)	0.6000000	0.0000000E+00
REL_ARC(5, 17)	0.8000000	0.0000000E+00
REL_ARC(5, 25)	0.5000000	0.0000000E+00
REL_ARC(6, 15)	0.8000000	0.0000000E+00
REL_ARC(7, 17)	0.5000000	0.0000000E+00
REL_ARC(7, 26)	0.6000000	0.0000000E+00
REL_ARC(7, 27)	0.6000000	0.0000000E+00
REL_ARC(8, 11)	0.5000000	0.0000000E+00
REL_ARC(9, 11)	0.7000000	0.0000000E+00
REL_ARC(10, 11)	0.5000000	0.0000000E+00
REL_ARC(11, 12)	0.5000000	0.0000000E+00
REL_ARC(11, 13)	0.7000000	0.0000000E+00
REL_ARC(11, 14)	0.7000000	0.0000000E+00
REL_ARC(12, 18)	0.5000000	0.0000000E+00
REL_ARC(12, 19)	0.5000000	0.0000000E+00
REL_ARC(12, 20)	0.5000000	0.0000000E+00
REL_ARC(13, 17)	0.7000000	0.0000000E+00
REL_ARC(13, 21)	0.5000000	0.0000000E+00
REL_ARC(14, 17)	0.7000000	0.0000000E+00

REL_ARC(15, 16)	0.8000000	0.0000000E+00
REL_ARC(16, 17)	0.8000000	0.0000000E+00
REL_ARC(17, 18)	0.6000000	0.0000000E+00
REL_ARC(17, 19)	0.6000000	0.0000000E+00
REL_ARC(17, 20)	0.6000000	0.0000000E+00
REL_ARC(17, 21)	0.6000000	0.0000000E+00
REL_ARC(17, 22)	0.6000000	0.0000000E+00
REL_ARC(17, 23)	0.6000000	0.0000000E+00
REL_ARC(17, 24)	0.6000000	0.0000000E+00
REL_ARC(17, 25)	0.6000000	0.0000000E+00
REL_ARC(17, 26)	0.6000000	0.0000000E+00
REL_ARC(17, 27)	0.6000000	0.0000000E+00
REL_ARC(18, 28)	1.000000	0.0000000E+00
REL_ARC(19, 28)	1.000000	0.0000000E+00
REL_ARC(20, 28)	1.000000	0.0000000E+00
REL_ARC(21, 28)	1.000000	0.0000000E+00
REL_ARC(22, 28)	1.000000	0.0000000E+00
REL_ARC(23, 28)	1.000000	0.0000000E+00
REL_ARC(24, 28)	1.000000	0.0000000E+00
REL_ARC(25, 28)	1.000000	0.0000000E+00
REL_ARC(26, 28)	1.000000	0.0000000E+00
REL_ARC(27, 28)	1.000000	0.0000000E+00
IMP_CAP(1, 2)	0.0000000E+00	0.0000000E+00
IMP_CAP(1, 3)	0.0000000E+00	0.0000000E+00
IMP_CAP(1, 4)	0.0000000E+00	0.0000000E+00
IMP_CAP(1, 5)	0.0000000E+00	0.0000000E+00
IMP_CAP(1, 6)	0.0000000E+00	0.0000000E+00
IMP_CAP(1, 7)	0.0000000E+00	0.0000000E+00
IMP_CAP(2, 8)	0.0000000E+00	1.181534
IMP_CAP(2, 9)	0.0000000E+00	1.334910
IMP_CAP(3, 9)	0.0000000E+00	5.834319
IMP_CAP(3, 10)	0.0000000E+00	5.939410
IMP_CAP(4, 8)	0.0000000E+00	1.581534
IMP_CAP(4, 10)	0.0000000E+00	5.939410
IMP_CAP(4, 17)	0.0000000E+00	1.158000
IMP_CAP(5, 17)	0.0000000E+00	1.544000
IMP_CAP(5, 25)	0.0000000E+00	4.300000
IMP_CAP(6, 15)	0.0000000E+00	9.600000
IMP_CAP(7, 17)	0.0000000E+00	9.565001
IMP_CAP(7, 26)	0.0000000E+00	0.6983377E-06
IMP_CAP(7, 27)	66.66667	0.0000000E+00
IMP_CAP(8, 11)	0.0000000E+00	9.600000
IMP_CAP(9, 11)	0.0000000E+00	9.600000
IMP_CAP(10, 11)	0.0000000E+00	19.20000
IMP_CAP(11, 12)	0.0000000E+00	9.576739
IMP_CAP(11, 13)	0.0000000E+00	9.600000
IMP_CAP(11, 14)	0.0000000E+00	9.600000
IMP_CAP(12, 18)	0.0000000E+00	9.600000
IMP_CAP(12, 19)	0.0000000E+00	0.6000000
IMP_CAP(12, 20)	0.0000000E+00	9.600000
IMP_CAP(13, 17)	0.0000000E+00	9.588589
IMP_CAP(13, 21)	0.0000000E+00	9.600000

IMP_CAP(14, 17)	0.0000000E+00	4.779276
IMP_CAP(15, 16)	0.0000000E+00	9.600000
IMP_CAP(16, 17)	0.0000000E+00	9.544001
IMP_CAP(17, 18)	0.0000000E+00	0.6000000
IMP_CAP(17, 19)	0.0000000E+00	0.6000000
IMP_CAP(17, 20)	0.0000000E+00	0.6000000
IMP_CAP(17, 21)	0.0000000E+00	0.6000000
IMP_CAP(17, 22)	0.0000000E+00	0.6000000
IMP_CAP(17, 23)	0.0000000E+00	0.6000000
IMP_CAP(17, 24)	0.0000000E+00	0.6000000
IMP_CAP(17, 25)	0.0000000E+00	0.6000000
IMP_CAP(17, 26)	0.0000000E+00	0.6000000
IMP_CAP(17, 27)	0.0000000E+00	0.6000000
IMP_CAP(18, 28)	0.0000000E+00	0.0000000E+00
IMP_CAP(19, 28)	0.0000000E+00	0.0000000E+00
IMP_CAP(20, 28)	0.0000000E+00	0.0000000E+00
IMP_CAP(21, 28)	0.0000000E+00	0.0000000E+00
IMP_CAP(22, 28)	0.0000000E+00	0.0000000E+00
IMP_CAP(23, 28)	0.0000000E+00	0.0000000E+00
IMP_CAP(24, 28)	0.0000000E+00	0.0000000E+00
IMP_CAP(25, 28)	0.0000000E+00	0.0000000E+00
IMP_CAP(26, 28)	0.0000000E+00	0.0000000E+00
IMP_CAP(27, 28)	0.0000000E+00	0.0000000E+00
IMP_REL_ARC(1, 2)	0.0000000E+00	43279.08
IMP_REL_ARC(1, 3)	0.0000000E+00	43279.08
IMP_REL_ARC(1, 4)	0.0000000E+00	43279.08
IMP_REL_ARC(1, 5)	0.0000000E+00	43279.08
IMP_REL_ARC(1, 6)	0.0000000E+00	43279.08
IMP_REL_ARC(1, 7)	0.0000000E+00	43279.08
IMP_REL_ARC(2, 8)	0.0000000E+00	34619.80
IMP_REL_ARC(2, 9)	0.0000000E+00	34556.99
IMP_REL_ARC(3, 9)	0.0000000E+00	21391.02
IMP_REL_ARC(3, 10)	0.0000000E+00	21548.65
IMP_REL_ARC(4, 8)	0.0000000E+00	34618.65
IMP_REL_ARC(4, 10)	0.0000000E+00	21548.65
IMP_REL_ARC(4, 17)	0.0000000E+00	25956.95
IMP_REL_ARC(5, 17)	0.0000000E+00	34609.27
IMP_REL_ARC(5, 25)	0.0000000E+00	21039.54
IMP_REL_ARC(6, 15)	0.0000000E+00	34623.27
IMP_REL_ARC(7, 17)	0.0000000E+00	21555.54
IMP_REL_ARC(7, 26)	0.0000000E+00	25892.45
IMP_REL_ARC(7, 27)	0.0000000E+00	25825.78
IMP_REL_ARC(8, 11)	0.0000000E+00	21639.54
IMP_REL_ARC(9, 11)	0.0000000E+00	30295.36
IMP_REL_ARC(10, 11)	0.0000000E+00	21639.54
IMP_REL_ARC(11, 12)	0.0000000E+00	21583.71
IMP_REL_ARC(11, 13)	0.0000000E+00	30295.36
IMP_REL_ARC(11, 14)	0.0000000E+00	30295.36
IMP_REL_ARC(12, 18)	0.0000000E+00	21639.54
IMP_REL_ARC(12, 19)	0.0000000E+00	21639.54
IMP_REL_ARC(12, 20)	0.0000000E+00	21639.54
IMP_REL_ARC(13, 17)	0.0000000E+00	30275.79

IMP_REL_ARC(13, 21)	0.0000000E+00	21639.54
IMP_REL_ARC(14, 17)	0.0000000E+00	30277.59
IMP_REL_ARC(15, 16)	0.0000000E+00	34623.27
IMP_REL_ARC(16, 17)	0.0000000E+00	34539.27
IMP_REL_ARC(17, 18)	0.0000000E+00	25967.45
IMP_REL_ARC(17, 19)	0.0000000E+00	25967.45
IMP_REL_ARC(17, 20)	0.0000000E+00	25967.45
IMP_REL_ARC(17, 21)	0.0000000E+00	25967.45
IMP_REL_ARC(17, 22)	0.0000000E+00	25967.45
IMP_REL_ARC(17, 23)	0.0000000E+00	25967.45
IMP_REL_ARC(17, 24)	0.0000000E+00	25967.45
IMP_REL_ARC(17, 25)	0.0000000E+00	25967.45
IMP_REL_ARC(17, 26)	0.0000000E+00	25967.45
IMP_REL_ARC(17, 27)	0.0000000E+00	25967.45
IMP_REL_ARC(18, 28)	0.0000000E+00	43279.08
IMP_REL_ARC(19, 28)	0.0000000E+00	43279.08
IMP_REL_ARC(20, 28)	0.0000000E+00	43279.08
IMP_REL_ARC(21, 28)	0.0000000E+00	43279.08
IMP_REL_ARC(22, 28)	0.0000000E+00	43279.08
IMP_REL_ARC(23, 28)	0.0000000E+00	43279.08
IMP_REL_ARC(24, 28)	0.0000000E+00	43279.08
IMP_REL_ARC(25, 28)	0.0000000E+00	43279.08
IMP_REL_ARC(26, 28)	0.0000000E+00	43279.08
IMP_REL_ARC(27, 28)	0.0000000E+00	43279.08
FLOW(1, 2)	164.8000	0.0000000E+00
FLOW(1, 3)	453.7500	0.0000000E+00
FLOW(1, 4)	91.45000	0.0000000E+00
FLOW(1, 5)	311.2000	0.0000000E+00
FLOW(1, 6)	67.20000	0.0000000E+00
FLOW(1, 7)	172.0000	0.0000000E+00
FLOW(2, 8)	4.800000	0.0000000E+00
FLOW(2, 9)	160.0000	0.0000000E+00
FLOW(3, 9)	375.0000	0.0000000E+00
FLOW(3, 10)	78.75000	0.0000000E+00
FLOW(4, 8)	6.400000	0.0000000E+00
FLOW(4, 10)	78.75000	0.0000000E+00
FLOW(4, 17)	6.300000	0.0000000E+00
FLOW(5, 17)	11.20000	0.0000000E+00
FLOW(5, 25)	300.0000	0.0000000E+00
FLOW(6, 15)	67.20000	0.0000000E+00
FLOW(7, 17)	42.00000	0.0000000E+00
FLOW(7, 26)	45.00000	0.0000000E+00
FLOW(7, 27)	85.00000	0.0000000E+00
FLOW(8, 11)	11.20000	0.0000000E+00
FLOW(9, 11)	535.0000	0.0000000E+00
FLOW(10, 11)	157.5000	0.0000000E+00
FLOW(11, 12)	66.00000	0.0000000E+00
FLOW(11, 13)	608.3000	0.0000000E+00
FLOW(11, 14)	29.40000	0.0000000E+00
FLOW(12, 18)	66.00000	0.0000000E+00
FLOW(12, 19)	0.0000000E+00	0.0000000E+00
FLOW(12, 20)	0.0000000E+00	0.0000000E+00

FLOW(13, 17)	58.80000	0.0000000E+00
FLOW(13, 21)	549.5000	0.0000000E+00
FLOW(14, 17)	29.40000	0.0000000E+00
FLOW(15, 16)	67.20000	0.0000000E+00
FLOW(16, 17)	67.20000	0.0000000E+00
FLOW(17, 18)	0.0000000E+00	0.0000000E+00
FLOW(17, 19)	0.0000000E+00	0.0000000E+00
FLOW(17, 20)	0.0000000E+00	0.0000000E+00
FLOW(17, 21)	42.44924	0.0000000E+00
FLOW(17, 22)	42.48359	0.0000000E+00
FLOW(17, 23)	42.48359	0.0000000E+00
FLOW(17, 24)	0.0000000E+00	0.0000000E+00
FLOW(17, 25)	42.48359	0.0000000E+00
FLOW(17, 26)	45.00000	0.0000000E+00
FLOW(17, 27)	0.0000000E+00	0.0000000E+00
FLOW(18, 28)	66.00000	0.0000000E+00
FLOW(19, 28)	0.0000000E+00	0.0000000E+00
FLOW(20, 28)	0.0000000E+00	0.0000000E+00
FLOW(21, 28)	591.9492	0.0000000E+00
FLOW(22, 28)	42.48359	0.0000000E+00
FLOW(23, 28)	42.48359	0.0000000E+00
FLOW(24, 28)	0.0000000E+00	0.0000000E+00
FLOW(25, 28)	342.4836	0.0000000E+00
FLOW(26, 28)	90.00000	0.0000000E+00
FLOW(27, 28)	85.00000	0.0000000E+00
Y_ARC(1, 2)	0.0000000E+00	187.5145
Y_ARC(1, 3)	0.0000000E+00	187.5145
Y_ARC(1, 4)	0.0000000E+00	187.5145
Y_ARC(1, 5)	0.0000000E+00	187.5145
Y_ARC(1, 6)	0.0000000E+00	187.5145
Y_ARC(1, 7)	0.0000000E+00	187.5145
Y_ARC(2, 8)	0.0000000E+00	187.5145
Y_ARC(2, 9)	0.0000000E+00	187.5145
Y_ARC(3, 9)	0.0000000E+00	187.5145
Y_ARC(3, 10)	0.0000000E+00	187.5145
Y_ARC(4, 8)	0.0000000E+00	187.5145
Y_ARC(4, 10)	0.0000000E+00	187.5145
Y_ARC(4, 17)	0.0000000E+00	187.5145
Y_ARC(5, 17)	0.0000000E+00	187.5145
Y_ARC(5, 25)	0.0000000E+00	187.5145
Y_ARC(6, 15)	0.0000000E+00	187.5145
Y_ARC(7, 17)	0.0000000E+00	187.5145
Y_ARC(7, 26)	0.0000000E+00	187.5145
Y_ARC(7, 27)	0.0000000E+00	187.5145
Y_ARC(8, 11)	0.0000000E+00	187.5145
Y_ARC(9, 11)	0.2990112	0.0000000E+00
Y_ARC(10, 11)	0.0000000E+00	187.5145
Y_ARC(11, 12)	0.0000000E+00	187.5145
Y_ARC(11, 13)	0.3834802	0.0000000E+00
Y_ARC(11, 14)	0.0000000E+00	187.5145
Y_ARC(12, 18)	0.0000000E+00	187.5145
Y_ARC(12, 19)	0.0000000E+00	187.5145

Y_ARC(12, 20)	0.0000000E+00	187.5145
Y_ARC(13, 17)	0.0000000E+00	187.5145
Y_ARC(13, 21)	0.3175086	0.0000000E+00
Y_ARC(14, 17)	0.0000000E+00	187.5145
Y_ARC(15, 16)	0.0000000E+00	187.5145
Y_ARC(16, 17)	0.0000000E+00	187.5145
Y_ARC(17, 18)	0.0000000E+00	187.5145
Y_ARC(17, 19)	0.0000000E+00	187.5145
Y_ARC(17, 20)	0.0000000E+00	187.5145
Y_ARC(17, 21)	0.0000000E+00	187.5145
Y_ARC(17, 22)	0.0000000E+00	187.5145
Y_ARC(17, 23)	0.0000000E+00	187.5145
Y_ARC(17, 24)	0.0000000E+00	187.5145
Y_ARC(17, 25)	0.0000000E+00	187.5145
Y_ARC(17, 26)	0.0000000E+00	187.5145
Y_ARC(17, 27)	0.0000000E+00	187.5145
Y_ARC(18, 28)	0.0000000E+00	187.5145
Y_ARC(19, 28)	0.0000000E+00	187.5145
Y_ARC(20, 28)	0.0000000E+00	187.5145
Y_ARC(21, 28)	0.0000000E+00	187.5145
Y_ARC(22, 28)	0.0000000E+00	187.5145
Y_ARC(23, 28)	0.0000000E+00	187.5145
Y_ARC(24, 28)	0.0000000E+00	187.5145
Y_ARC(25, 28)	0.0000000E+00	187.5145
Y_ARC(26, 28)	0.0000000E+00	187.5145
Y_ARC(27, 28)	0.0000000E+00	187.5145

Row	Slack or Surplus	Dual Price
1	885.3710	1.000000
2	0.0000000E+00	0.0000000E+00
3	0.3000000	0.0000000E+00
4	0.8500000	0.0000000E+00
5	0.9700000	0.0000000E+00
6	0.0000000E+00	0.0000000E+00
7	0.0000000E+00	0.0000000E+00
8	0.9600000	0.0000000E+00
9	0.9600000	0.0000000E+00
10	0.0000000E+00	0.0000000E+00
11	0.7900000	0.0000000E+00
12	0.3000000	0.0000000E+00
13	0.8900000	0.0000000E+00
14	0.0000000E+00	0.0000000E+00
15	0.9400000	0.0000000E+00
16	0.9100000	0.0000000E+00
17	0.8200000	0.0000000E+00
18	0.9300000	0.0000000E+00
19	0.0000000E+00	0.0000000E+00
20	0.0000000E+00	0.0000000E+00
21	0.0000000E+00	0.0000000E+00
22	0.0000000E+00	0.0000000E+00
23	0.0000000E+00	0.0000000E+00
24	0.0000000E+00	0.0000000E+00

25	0.0000000E+00	0.0000000E+00
26	0.0000000E+00	0.0000000E+00
27	0.0000000E+00	0.0000000E+00
28	0.0000000E+00	0.0000000E+00
29	0.0000000E+00	0.0000000E+00
30	0.0000000E+00	0.0000000E+00
31	0.0000000E+00	0.0000000E+00
32	0.0000000E+00	0.0000000E+00
33	0.0000000E+00	0.0000000E+00
34	0.0000000E+00	0.0000000E+00
35	0.0000000E+00	0.0000000E+00
36	0.2000000	0.0000000E+00
37	0.2000000	0.0000000E+00
38	0.5000000	0.0000000E+00
39	0.5000000	0.0000000E+00
40	0.2000000	0.0000000E+00
41	0.5000000	0.0000000E+00
42	0.4000000	0.0000000E+00
43	0.2000000	0.0000000E+00
44	0.5000000	0.0000000E+00
45	0.2000000	0.0000000E+00
46	0.5000000	0.0000000E+00
47	0.4000000	0.0000000E+00
48	0.4000000	0.0000000E+00
49	0.5000000	0.0000000E+00
50	0.3000000	0.0000000E+00
51	0.5000000	0.0000000E+00
52	0.5000000	0.0000000E+00
53	0.3000000	0.0000000E+00
54	0.3000000	0.0000000E+00
55	0.5000000	0.0000000E+00
56	0.5000000	0.0000000E+00
57	0.5000000	0.0000000E+00
58	0.3000000	0.0000000E+00
59	0.5000000	0.0000000E+00
60	0.3000000	0.0000000E+00
61	0.2000000	0.0000000E+00
62	0.2000000	0.0000000E+00
63	0.4000000	0.0000000E+00
64	0.4000000	0.0000000E+00
65	0.4000000	0.0000000E+00
66	0.4000000	0.0000000E+00
67	0.4000000	0.0000000E+00
68	0.4000000	0.0000000E+00
69	0.4000000	0.0000000E+00
70	0.4000000	0.0000000E+00
71	0.4000000	0.0000000E+00
72	0.4000000	0.0000000E+00
73	0.0000000E+00	0.0000000E+00
74	0.0000000E+00	0.0000000E+00
75	0.0000000E+00	0.0000000E+00
76	0.0000000E+00	0.0000000E+00

77	0.0000000E+00	0.0000000E+00
78	0.0000000E+00	0.0000000E+00
79	0.0000000E+00	0.0000000E+00
80	0.0000000E+00	0.0000000E+00
81	0.0000000E+00	0.0000000E+00
82	0.0000000E+00	0.0000000E+00
83	0.0000000E+00	0.5770544
84	0.0000000E+00	0.3313618
85	0.0000000E+00	0.3313618
86	0.0000000E+00	0.5770544
87	0.0000000E+00	0.5770544
88	0.0000000E+00	0.5770544
89	0.0000000E+00	1.000000
90	0.0000000E+00	1.000000
91	0.0000000E+00	1.000000
92	19.20000	0.0000000E+00
93	0.0000000E+00	1.000000
94	0.0000000E+00	1.000000
95	0.0000000E+00	1.000000
96	408.8000	0.0000000E+00
97	53.00000	0.0000000E+00
98	682.5000	0.0000000E+00
99	0.0000000E+00	0.4229456
100	231.7000	0.0000000E+00
101	21.00000	0.0000000E+00
102	534.0000	0.0000000E+00
103	37.50000	0.0000000E+00
104	600.0000	0.0000000E+00
105	0.0000000E+00	0.2328972
106	50.50000	0.0000000E+00
107	0.0000000E+00	0.4229456
108	105.6000	0.0000000E+00
109	0.0000000E+00	1.000000
110	45.00000	0.0000000E+00
111	45.00000	0.0000000E+00
112	45.00000	0.0000000E+00
113	2.550756	0.0000000E+00
114	2.516415	0.0000000E+00
115	2.516415	0.0000000E+00
116	45.00000	0.0000000E+00
117	2.516415	0.0000000E+00
118	0.0000000E+00	0.0000000E+00
119	45.00000	0.0000000E+00
120	370.2290	0.0000000E+00
121	215.0290	0.0000000E+00
122	0.2900935E-01	0.0000000E+00
123	296.2790	0.0000000E+00
124	368.6290	0.0000000E+00
125	296.2790	0.0000000E+00
126	368.7290	0.0000000E+00
127	363.8290	0.0000000E+00
128	75.02901	0.0000000E+00

129	307.8290	0.0000000E+00
130	333.0290	0.0000000E+00
131	330.0290	0.0000000E+00
132	290.0290	0.0000000E+00
133	363.8290	0.0000000E+00
134	0.0000000E+00	0.3504944
135	217.5290	0.0000000E+00
136	309.0290	0.0000000E+00
137	0.0000000E+00	0.3082599
138	345.6290	0.0000000E+00
139	309.0290	0.0000000E+00
140	375.0290	0.0000000E+00
141	375.0290	0.0000000E+00
142	316.2290	0.0000000E+00
143	0.0000000E+00	0.3412457
144	345.6290	0.0000000E+00
145	307.8290	0.0000000E+00
146	307.8290	0.0000000E+00
147	375.0290	0.0000000E+00
148	375.0290	0.0000000E+00
149	375.0290	0.0000000E+00
150	332.5798	0.0000000E+00
151	332.5454	0.0000000E+00
152	332.5454	0.0000000E+00
153	375.0290	0.0000000E+00
154	332.5454	0.0000000E+00
155	330.0290	0.0000000E+00
156	375.0290	0.0000000E+00
157	0.0000000E+00	0.8000000E-02
158	0.0000000E+00	43279.08
159	0.0000000E+00	43279.08
160	0.0000000E+00	0.8000000E-02
161	0.0000000E+00	187.5145
162	0.0000000E+00	-1.000000
163	0.0000000E+00	-1.000000
164	0.0000000E+00	-1.000000
165	0.0000000E+00	-1.000000
166	0.0000000E+00	-1.000000
167	0.0000000E+00	-1.000000
168	0.0000000E+00	-1.000000
169	0.0000000E+00	-0.4229456
170	0.0000000E+00	-0.6686382
171	0.0000000E+00	-0.4229456
172	0.0000000E+00	-0.4229456
173	0.0000000E+00	0.0000000E+00
174	0.0000000E+00	-0.2328972
175	0.0000000E+00	-0.4229456
176	0.0000000E+00	-1.000000
177	0.0000000E+00	-1.000000
178	0.0000000E+00	0.0000000E+00
179	0.0000000E+00	0.0000000E+00
180	0.0000000E+00	0.0000000E+00

181	0.0000000E+00	0.0000000E+00
182	0.0000000E+00	0.0000000E+00
183	0.0000000E+00	0.0000000E+00
184	0.0000000E+00	0.0000000E+00
185	0.0000000E+00	0.0000000E+00
186	0.0000000E+00	0.0000000E+00
187	0.0000000E+00	0.0000000E+00
188	0.0000000E+00	0.0000000E+00
189	0.0000000E+00	0.0000000E+00

```
MODEL:

! Model C for Network C;

! Since LINGO numbers from 1 rather than 0, subtract 1 from variables to
translate back to diagram.  For example, arc(1,2) in LINGO program = arc
(s,1) actual.;


SETS:
      NODE / 1..41 / : rel_node, imp_rel_node;
      ARC(NODE, NODE) / 1,2 1,3 1,4 1,5 1,6 1,7 2,12 3,12 4,8 5,12 6,9
7,10 8,11
            9,12 10,12 11,12 12,13 12,24 12,25 12,39 12,40 13,14 13,15
13,16
            13,17 13,18 13,19 13,20 13,21 13,22 13,23 14,24 14,25 14,26
14,27
            14,28 14,29 14,30 15,24 15,25 15,26 15,27 15,28 15,29 15,30
16,32 16,33
            17,31 17,32 17,33 17,34 17,37 18,31 18,34 18,35 19,39 20,40
21,36
            22,37 23,38 24,41 25,41 26,41 27,41 28,41 29,41 30,41 31,41
32,41
            33,41 34,41 35,41 36,41 37,41 38,41 39,41 40,41/ :
      cap, rel_arc, imp_cap, imp_rel_arc, flow, y_arc;
ENDSETS

DATA:

      rel_node = 1 .3 .7 .5 .8 1 .3 .7 .5 .8 1 .7 .7 .5 .8 1 .3 .7 .5 .5
1 .3 .7
            .7 .8 1 .3 .7 .5 .8 1 .3 .7 .5 .8 1 .3 .7 .5 .5 1;
      rel_arc = 1 1 1 1 1 1 .9 1 1 .6 .3 .6 1 .9 1 .7 .9 .6 .3 .6 .7
            1 1 .6 .3 .6 .7 .9 1 1 .6 .3 .6 .7 .9 1 1 .3 .6 .3 .6 .7 .9
1 1
            .6 .3 .6 .7 .9 1 1 .6 .3 .6 .7 .9 1 1 .6
            1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1;
      cap = 0 0 0 0 0 0
            1200 1200 300 1200 1200 1200 300 1200 1200 300 9600 75 75
1200 1200
            4800 4800 4800 4800 4800 2400 4800 4800 4800 2400
            4800 4800 2400 1200 1200 1200 1200 4800 4800 2400 1200 1200
1200 1200
            2400 1200 300 2400 1200 300 2400 1200 300 300 1200 1200 2400
2400 600
            0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
ENDDATA

MAX = Vef;

@FOR(NODE(I) : rel_node(I) + imp_rel_node(I) <= 1);
```

```
@FOR(ARC(I,J) : rel_arc(I,J) + imp_rel_arc(I,J) <= 1);

@FOR(ARC(I,J) | I #NE# 1 #AND# J #NE# 41 :
      flow(I,J) - ((cap(I,J) + imp_cap(I,J))*(rel_arc(I,J) +
imp_rel_arc(I,J))
      *(rel_node(J) + imp_rel_node(J))) <= 0);


@FOR(ARC(I,J) | I #NE# 1 #AND# J #NE# 41 :
      Vef - V + flow(I,J)*(1 - y_arc(I,J)) <= 0);

@SUM(ARC(I,J) | I #NE# 1 #AND# J #NE# 41 :
      cap(I,J)*imp_cap(I,J)) <= Bcap;

@SUM(ARC(I,J) : rel_arc(I,J)*imp_rel_arc(I,J))
      + @SUM(NODE(K):rel_node(K)*imp_rel_node(K)) <= Brel;

Bcap = 500000;

Brel = 0.1;

@SUM(ARC(I,J) : y_arc(I,J)) <= 1;

@FOR(NODE(J) | J #EQ# 1 :
      @SUM(ARC(J,K):flow(J,K)) - V = 0);

@FOR(NODE(J) | J #GT# 1 #AND# J #LT# 41 :
      @SUM(ARC(J,K):flow(J,K)) - @SUM(ARC(I,J):flow(I,J)) = 0);

@FOR(NODE(J) | J #EQ# 41 :
      V - @SUM(ARC(I,J):flow(I,J)) = 0);

END
```

OUTPUT FOR NETWORK C: MODEL C

Rows= 271 Vars= 351 No. integer vars= 0
Nonlinear rows= 108 Nonlinear vars= 249 Nonlinear constraints= 108
Nonzeros= 1083 Constraint nonz= 955 Density=0.011

Optimal solution found at step: 62
Objective value: 3077.901

Variable	Value	Reduced Cost
VEF	3077.901	0.0000000E+00
V	3907.022	0.0000000E+00
BCAP	500000.0	0.0000000E+00
BREL	0.1000000	0.0000000E+00
REL_NODE(1)	1.000000	0.0000000E+00
REL_NODE(2)	0.3000000	0.0000000E+00
REL_NODE(3)	0.7000000	0.0000000E+00
REL_NODE(4)	0.5000000	0.0000000E+00
REL_NODE(5)	0.8000000	0.0000000E+00
REL_NODE(6)	1.000000	0.0000000E+00
REL_NODE(7)	0.3000000	0.0000000E+00
REL_NODE(8)	0.7000000	0.0000000E+00
REL_NODE(9)	0.5000000	0.0000000E+00
REL_NODE(10)	0.8000000	0.0000000E+00
REL_NODE(11)	1.000000	0.0000000E+00
REL_NODE(12)	0.7000000	0.0000000E+00
REL_NODE(13)	0.7000000	0.0000000E+00
REL_NODE(14)	0.5000000	0.0000000E+00
REL_NODE(15)	0.8000000	0.0000000E+00
REL_NODE(16)	1.000000	0.0000000E+00
REL_NODE(17)	0.3000000	0.0000000E+00
REL_NODE(18)	0.7000000	0.0000000E+00
REL_NODE(19)	0.5000000	0.0000000E+00
REL_NODE(20)	0.5000000	0.0000000E+00
REL_NODE(21)	1.000000	0.0000000E+00
REL_NODE(22)	0.3000000	0.0000000E+00
REL_NODE(23)	0.7000000	0.0000000E+00
REL_NODE(24)	0.7000000	0.0000000E+00
REL_NODE(25)	0.8000000	0.0000000E+00
REL_NODE(26)	1.000000	0.0000000E+00
REL_NODE(27)	0.3000000	0.0000000E+00
REL_NODE(28)	0.7000000	0.0000000E+00
REL_NODE(29)	0.5000000	0.0000000E+00
REL_NODE(30)	0.8000000	0.0000000E+00
REL_NODE(31)	1.000000	0.0000000E+00
REL_NODE(32)	0.3000000	0.0000000E+00
REL_NODE(33)	0.7000000	0.0000000E+00
REL_NODE(34)	0.5000000	0.0000000E+00
REL_NODE(35)	0.8000000	0.0000000E+00
REL_NODE(36)	1.000000	0.0000000E+00

REL_NODE(37)	0.3000000	0.0000000E+00
REL_NODE(38)	0.7000000	0.0000000E+00
REL_NODE(39)	0.5000000	0.0000000E+00
REL_NODE(40)	0.5000000	0.0000000E+00
REL_NODE(41)	1.000000	0.0000000E+00
IMP_REL_NODE(1)	0.0000000E+00	3332.100
IMP_REL_NODE(2)	0.0000000E+00	999.6301
IMP_REL_NODE(3)	0.0000000E+00	2332.470
IMP_REL_NODE(4)	0.0000000E+00	1666.050
IMP_REL_NODE(5)	0.0000000E+00	2665.680
IMP_REL_NODE(6)	0.0000000E+00	3332.100
IMP_REL_NODE(7)	0.0000000E+00	999.6301
IMP_REL_NODE(8)	0.0000000E+00	2035.584
IMP_REL_NODE(9)	0.0000000E+00	1319.115
IMP_REL_NODE(10)	0.0000000E+00	1971.810
IMP_REL_NODE(11)	0.0000000E+00	3186.626
IMP_REL_NODE(12)	0.1428571	0.0000000E+00
IMP_REL_NODE(13)	0.0000000E+00	2332.470
IMP_REL_NODE(14)	0.0000000E+00	1666.050
IMP_REL_NODE(15)	0.0000000E+00	2665.680
IMP_REL_NODE(16)	0.0000000E+00	3332.100
IMP_REL_NODE(17)	0.0000000E+00	999.6301
IMP_REL_NODE(18)	0.0000000E+00	2332.470
IMP_REL_NODE(19)	0.0000000E+00	1666.050
IMP_REL_NODE(20)	0.0000000E+00	1666.050
IMP_REL_NODE(21)	0.0000000E+00	3332.100
IMP_REL_NODE(22)	0.0000000E+00	999.6301
IMP_REL_NODE(23)	0.0000000E+00	2332.470
IMP_REL_NODE(24)	0.0000000E+00	2330.837
IMP_REL_NODE(25)	0.0000000E+00	2664.864
IMP_REL_NODE(26)	0.0000000E+00	3332.100
IMP_REL_NODE(27)	0.0000000E+00	999.6301
IMP_REL_NODE(28)	0.0000000E+00	2332.470
IMP_REL_NODE(29)	0.0000000E+00	1666.050
IMP_REL_NODE(30)	0.0000000E+00	2665.680
IMP_REL_NODE(31)	0.0000000E+00	3332.100
IMP_REL_NODE(32)	0.0000000E+00	999.6301
IMP_REL_NODE(33)	0.0000000E+00	2332.470
IMP_REL_NODE(34)	0.0000000E+00	1666.050
IMP_REL_NODE(35)	0.0000000E+00	2665.680
IMP_REL_NODE(36)	0.0000000E+00	3332.100
IMP_REL_NODE(37)	0.0000000E+00	999.6301
IMP_REL_NODE(38)	0.0000000E+00	2332.470
IMP_REL_NODE(39)	0.0000000E+00	1639.920
IMP_REL_NODE(40)	0.0000000E+00	1635.565
IMP_REL_NODE(41)	0.0000000E+00	3332.100
CAP(1, 2)	0.0000000E+00	0.0000000E+00
CAP(1, 3)	0.0000000E+00	0.0000000E+00
CAP(1, 4)	0.0000000E+00	0.0000000E+00
CAP(1, 5)	0.0000000E+00	0.0000000E+00
CAP(1, 6)	0.0000000E+00	0.0000000E+00
CAP(1, 7)	0.0000000E+00	0.0000000E+00

CAP(2, 12)	1200.000	0.0000000E+00
CAP(3, 12)	1200.000	0.0000000E+00
CAP(4, 8)	300.0000	0.0000000E+00
CAP(5, 12)	1200.000	0.0000000E+00
CAP(6, 9)	1200.000	0.0000000E+00
CAP(7, 10)	1200.000	0.0000000E+00
CAP(8, 11)	300.0000	0.0000000E+00
CAP(9, 12)	1200.000	0.0000000E+00
CAP(10, 12)	1200.000	0.0000000E+00
CAP(11, 12)	300.0000	0.0000000E+00
CAP(12, 13)	9600.000	0.0000000E+00
CAP(12, 24)	75.00000	0.0000000E+00
CAP(12, 25)	75.00000	0.0000000E+00
CAP(12, 39)	1200.000	0.0000000E+00
CAP(12, 40)	1200.000	0.0000000E+00
CAP(13, 14)	4800.000	0.0000000E+00
CAP(13, 15)	4800.000	0.0000000E+00
CAP(13, 16)	4800.000	0.0000000E+00
CAP(13, 17)	4800.000	0.0000000E+00
CAP(13, 18)	4800.000	0.0000000E+00
CAP(13, 19)	2400.000	0.0000000E+00
CAP(13, 20)	4800.000	0.0000000E+00
CAP(13, 21)	4800.000	0.0000000E+00
CAP(13, 22)	4800.000	0.0000000E+00
CAP(13, 23)	2400.000	0.0000000E+00
CAP(14, 24)	4800.000	0.0000000E+00
CAP(14, 25)	4800.000	0.0000000E+00
CAP(14, 26)	2400.000	0.0000000E+00
CAP(14, 27)	1200.000	0.0000000E+00
CAP(14, 28)	1200.000	0.0000000E+00
CAP(14, 29)	1200.000	0.0000000E+00
CAP(14, 30)	1200.000	0.0000000E+00
CAP(15, 24)	4800.000	0.0000000E+00
CAP(15, 25)	4800.000	0.0000000E+00
CAP(15, 26)	2400.000	0.0000000E+00
CAP(15, 27)	1200.000	0.0000000E+00
CAP(15, 28)	1200.000	0.0000000E+00
CAP(15, 29)	1200.000	0.0000000E+00
CAP(15, 30)	1200.000	0.0000000E+00
CAP(16, 32)	2400.000	0.0000000E+00
CAP(16, 33)	1200.000	0.0000000E+00
CAP(17, 31)	300.0000	0.0000000E+00
CAP(17, 32)	2400.000	0.0000000E+00
CAP(17, 33)	1200.000	0.0000000E+00
CAP(17, 34)	300.0000	0.0000000E+00
CAP(17, 37)	2400.000	0.0000000E+00
CAP(18, 31)	1200.000	0.0000000E+00
CAP(18, 34)	300.0000	0.0000000E+00
CAP(18, 35)	300.0000	0.0000000E+00
CAP(19, 39)	1200.000	0.0000000E+00
CAP(20, 40)	1200.000	0.0000000E+00
CAP(21, 36)	2400.000	0.0000000E+00

CAP(22, 37)	2400.000	0.0000000E+00
CAP(23, 38)	600.0000	0.0000000E+00
CAP(24, 41)	0.0000000E+00	0.0000000E+00
CAP(25, 41)	0.0000000E+00	0.0000000E+00
CAP(26, 41)	0.0000000E+00	0.0000000E+00
CAP(27, 41)	0.0000000E+00	0.0000000E+00
CAP(28, 41)	0.0000000E+00	0.0000000E+00
CAP(29, 41)	0.0000000E+00	0.0000000E+00
CAP(30, 41)	0.0000000E+00	0.0000000E+00
CAP(31, 41)	0.0000000E+00	0.0000000E+00
CAP(32, 41)	0.0000000E+00	0.0000000E+00
CAP(33, 41)	0.0000000E+00	0.0000000E+00
CAP(34, 41)	0.0000000E+00	0.0000000E+00
CAP(35, 41)	0.0000000E+00	0.0000000E+00
CAP(36, 41)	0.0000000E+00	0.0000000E+00
CAP(37, 41)	0.0000000E+00	0.0000000E+00
CAP(38, 41)	0.0000000E+00	0.0000000E+00
CAP(39, 41)	0.0000000E+00	0.0000000E+00
CAP(40, 41)	0.0000000E+00	0.0000000E+00
REL_ARC(1, 2)	1.000000	0.0000000E+00
REL_ARC(1, 3)	1.000000	0.0000000E+00
REL_ARC(1, 4)	1.000000	0.0000000E+00
REL_ARC(1, 5)	1.000000	0.0000000E+00
REL_ARC(1, 6)	1.000000	0.0000000E+00
REL_ARC(1, 7)	1.000000	0.0000000E+00
REL_ARC(2, 12)	0.9000000	0.0000000E+00
REL_ARC(3, 12)	1.000000	0.0000000E+00
REL_ARC(4, 8)	1.000000	0.0000000E+00
REL_ARC(5, 12)	0.6000000	0.0000000E+00
REL_ARC(6, 9)	0.3000000	0.0000000E+00
REL_ARC(7, 10)	0.6000000	0.0000000E+00
REL_ARC(8, 11)	1.000000	0.0000000E+00
REL_ARC(9, 12)	0.9000000	0.0000000E+00
REL_ARC(10, 12)	1.000000	0.0000000E+00
REL_ARC(11, 12)	0.7000000	0.0000000E+00
REL_ARC(12, 13)	0.9000000	0.0000000E+00
REL_ARC(12, 24)	0.6000000	0.0000000E+00
REL_ARC(12, 25)	0.3000000	0.0000000E+00
REL_ARC(12, 39)	0.6000000	0.0000000E+00
REL_ARC(12, 40)	0.7000000	0.0000000E+00
REL_ARC(13, 14)	1.000000	0.0000000E+00
REL_ARC(13, 15)	1.000000	0.0000000E+00
REL_ARC(13, 16)	0.6000000	0.0000000E+00
REL_ARC(13, 17)	0.3000000	0.0000000E+00
REL_ARC(13, 18)	0.6000000	0.0000000E+00
REL_ARC(13, 19)	0.7000000	0.0000000E+00
REL_ARC(13, 20)	0.9000000	0.0000000E+00
REL_ARC(13, 21)	1.000000	0.0000000E+00
REL_ARC(13, 22)	1.000000	0.0000000E+00
REL_ARC(13, 23)	0.6000000	0.0000000E+00
REL_ARC(14, 24)	0.3000000	0.0000000E+00
REL_ARC(14, 25)	0.6000000	0.0000000E+00

REL_ARC(14, 26)	0.7000000	0.0000000E+00
REL_ARC(14, 27)	0.9000000	0.0000000E+00
REL_ARC(14, 28)	1.000000	0.0000000E+00
REL_ARC(14, 29)	1.000000	0.0000000E+00
REL_ARC(14, 30)	0.3000000	0.0000000E+00
REL_ARC(15, 24)	0.6000000	0.0000000E+00
REL_ARC(15, 25)	0.3000000	0.0000000E+00
REL_ARC(15, 26)	0.6000000	0.0000000E+00
REL_ARC(15, 27)	0.7000000	0.0000000E+00
REL_ARC(15, 28)	0.9000000	0.0000000E+00
REL_ARC(15, 29)	1.000000	0.0000000E+00
REL_ARC(15, 30)	1.000000	0.0000000E+00
REL_ARC(16, 32)	0.6000000	0.0000000E+00
REL_ARC(16, 33)	0.3000000	0.0000000E+00
REL_ARC(17, 31)	0.6000000	0.0000000E+00
REL_ARC(17, 32)	0.7000000	0.0000000E+00
REL_ARC(17, 33)	0.9000000	0.0000000E+00
REL_ARC(17, 34)	1.000000	0.0000000E+00
REL_ARC(17, 37)	1.000000	0.0000000E+00
REL_ARC(18, 31)	0.6000000	0.0000000E+00
REL_ARC(18, 34)	0.3000000	0.0000000E+00
REL_ARC(18, 35)	0.6000000	0.0000000E+00
REL_ARC(19, 39)	0.7000000	0.0000000E+00
REL_ARC(20, 40)	0.9000000	0.0000000E+00
REL_ARC(21, 36)	1.000000	0.0000000E+00
REL_ARC(22, 37)	1.000000	0.0000000E+00
REL_ARC(23, 38)	0.6000000	0.0000000E+00
REL_ARC(24, 41)	1.000000	0.0000000E+00
REL_ARC(25, 41)	1.000000	0.0000000E+00
REL_ARC(26, 41)	1.000000	0.0000000E+00
REL_ARC(27, 41)	1.000000	0.0000000E+00
REL_ARC(28, 41)	1.000000	0.0000000E+00
REL_ARC(29, 41)	1.000000	0.0000000E+00
REL_ARC(30, 41)	1.000000	0.0000000E+00
REL_ARC(31, 41)	1.000000	0.0000000E+00
REL_ARC(32, 41)	1.000000	0.0000000E+00
REL_ARC(33, 41)	1.000000	0.0000000E+00
REL_ARC(34, 41)	1.000000	0.0000000E+00
REL_ARC(35, 41)	1.000000	0.0000000E+00
REL_ARC(36, 41)	1.000000	0.0000000E+00
REL_ARC(37, 41)	1.000000	0.0000000E+00
REL_ARC(38, 41)	1.000000	0.0000000E+00
REL_ARC(39, 41)	1.000000	0.0000000E+00
REL_ARC(40, 41)	1.000000	0.0000000E+00
IMP_CAP(1, 2)	0.0000000E+00	0.0000000E+00
IMP_CAP(1, 3)	0.0000000E+00	0.0000000E+00
IMP_CAP(1, 4)	0.0000000E+00	0.0000000E+00
IMP_CAP(1, 5)	0.0000000E+00	0.0000000E+00
IMP_CAP(1, 6)	0.0000000E+00	0.0000000E+00
IMP_CAP(1, 7)	0.0000000E+00	0.0000000E+00
IMP_CAP(2, 12)	0.0000000E+00	0.5184743
IMP_CAP(3, 12)	0.0000000E+00	0.4057806

IMP_CAP(4, 8)	589.2151	0.0000000E+00
IMP_CAP(5, 12)	0.0000000E+00	0.4474882
IMP_CAP(6, 9)	0.0000000E+00	0.7902936
IMP_CAP(7, 10)	0.0000000E+00	0.4722709
IMP_CAP(8, 11)	322.4506	0.0000000E+00
IMP_CAP(9, 12)	0.0000000E+00	0.9348483
IMP_CAP(10, 12)	0.0000000E+00	0.9348483
IMP_CAP(11, 12)	755.0010	0.0000000E+00
IMP_CAP(12, 13)	0.0000000E+00	7.478787
IMP_CAP(12, 24)	0.0000000E+00	0.4318565E-01
IMP_CAP(12, 25)	0.0000000E+00	0.4971810E-01
IMP_CAP(12, 39)	0.0000000E+00	0.9239610
IMP_CAP(12, 40)	0.0000000E+00	0.9221465
IMP_CAP(13, 14)	0.0000000E+00	3.739393
IMP_CAP(13, 15)	0.0000000E+00	3.739393
IMP_CAP(13, 16)	0.0000000E+00	3.739393
IMP_CAP(13, 17)	0.0000000E+00	3.739393
IMP_CAP(13, 18)	0.0000000E+00	3.739393
IMP_CAP(13, 19)	0.0000000E+00	1.869697
IMP_CAP(13, 20)	0.0000000E+00	3.739393
IMP_CAP(13, 21)	0.0000000E+00	3.739393
IMP_CAP(13, 22)	0.0000000E+00	3.739393
IMP_CAP(13, 23)	0.0000000E+00	1.869697
IMP_CAP(14, 24)	0.0000000E+00	3.739393
IMP_CAP(14, 25)	0.0000000E+00	3.739393
IMP_CAP(14, 26)	0.0000000E+00	1.869697
IMP_CAP(14, 27)	0.0000000E+00	0.9348483
IMP_CAP(14, 28)	0.0000000E+00	0.9348483
IMP_CAP(14, 29)	0.0000000E+00	0.9348483
IMP_CAP(14, 30)	0.0000000E+00	0.9348483
IMP_CAP(15, 24)	0.0000000E+00	3.739393
IMP_CAP(15, 25)	0.0000000E+00	3.739393
IMP_CAP(15, 26)	0.0000000E+00	1.869697
IMP_CAP(15, 27)	0.0000000E+00	0.9348483
IMP_CAP(15, 28)	0.0000000E+00	0.9348483
IMP_CAP(15, 29)	0.0000000E+00	0.9348483
IMP_CAP(15, 30)	0.0000000E+00	0.9348483
IMP_CAP(16, 32)	0.0000000E+00	1.869697
IMP_CAP(16, 33)	0.0000000E+00	0.9348483
IMP_CAP(17, 31)	0.0000000E+00	0.2337121
IMP_CAP(17, 32)	0.0000000E+00	1.869697
IMP_CAP(17, 33)	0.0000000E+00	0.9348483
IMP_CAP(17, 34)	0.0000000E+00	0.2337121
IMP_CAP(17, 37)	0.0000000E+00	1.869697
IMP_CAP(18, 31)	0.0000000E+00	0.9348483
IMP_CAP(18, 34)	0.0000000E+00	0.2337121
IMP_CAP(18, 35)	0.0000000E+00	0.2337121
IMP_CAP(19, 39)	0.0000000E+00	0.9348483
IMP_CAP(20, 40)	0.0000000E+00	0.9348483
IMP_CAP(21, 36)	0.0000000E+00	1.869697
IMP_CAP(22, 37)	0.0000000E+00	1.869697
IMP_CAP(23, 38)	0.0000000E+00	0.4674242

IMP_CAP(24, 41)	0.0000000E+00	0.0000000E+00
IMP_CAP(25, 41)	0.0000000E+00	0.0000000E+00
IMP_CAP(26, 41)	0.0000000E+00	0.0000000E+00
IMP_CAP(27, 41)	0.0000000E+00	0.0000000E+00
IMP_CAP(28, 41)	0.0000000E+00	0.0000000E+00
IMP_CAP(29, 41)	0.0000000E+00	0.0000000E+00
IMP_CAP(30, 41)	0.0000000E+00	0.0000000E+00
IMP_CAP(31, 41)	0.0000000E+00	0.0000000E+00
IMP_CAP(32, 41)	0.0000000E+00	0.0000000E+00
IMP_CAP(33, 41)	0.0000000E+00	0.0000000E+00
IMP_CAP(34, 41)	0.0000000E+00	0.0000000E+00
IMP_CAP(35, 41)	0.0000000E+00	0.0000000E+00
IMP_CAP(36, 41)	0.0000000E+00	0.0000000E+00
IMP_CAP(37, 41)	0.0000000E+00	0.0000000E+00
IMP_CAP(38, 41)	0.0000000E+00	0.0000000E+00
IMP_CAP(39, 41)	0.0000000E+00	0.0000000E+00
IMP_CAP(40, 41)	0.0000000E+00	0.0000000E+00
IMP_REL_ARC(1, 2)	0.0000000E+00	3332.100
IMP_REL_ARC(1, 3)	0.0000000E+00	3332.100
IMP_REL_ARC(1, 4)	0.0000000E+00	3332.100
IMP_REL_ARC(1, 5)	0.0000000E+00	3332.100
IMP_REL_ARC(1, 6)	0.0000000E+00	3332.100
IMP_REL_ARC(1, 7)	0.0000000E+00	3332.100
IMP_REL_ARC(2, 12)	0.0000000E+00	2443.720
IMP_REL_ARC(3, 12)	0.0000000E+00	2697.215
IMP_REL_ARC(4, 8)	0.0000000E+00	3124.280
IMP_REL_ARC(5, 12)	0.0000000E+00	1024.538
IMP_REL_ARC(6, 9)	0.0000000E+00	421.4049
IMP_REL_ARC(7, 10)	0.0000000E+00	1074.100
IMP_REL_ARC(8, 11)	0.0000000E+00	3186.626
IMP_REL_ARC(9, 12)	0.0000000E+00	2998.890
IMP_REL_ARC(10, 12)	0.0000000E+00	3332.100
IMP_REL_ARC(11, 12)	0.0000000E+00	1980.232
IMP_REL_ARC(12, 13)	0.0000000E+00	2998.890
IMP_REL_ARC(12, 24)	0.0000000E+00	1997.355
IMP_REL_ARC(12, 25)	0.0000000E+00	997.4526
IMP_REL_ARC(12, 39)	0.0000000E+00	1977.485
IMP_REL_ARC(12, 40)	0.0000000E+00	2310.695
IMP_REL_ARC(13, 14)	0.0000000E+00	3332.100
IMP_REL_ARC(13, 15)	0.0000000E+00	3332.100
IMP_REL_ARC(13, 16)	0.0000000E+00	1999.260
IMP_REL_ARC(13, 17)	0.0000000E+00	999.6301
IMP_REL_ARC(13, 18)	0.0000000E+00	1999.260
IMP_REL_ARC(13, 19)	0.0000000E+00	2332.470
IMP_REL_ARC(13, 20)	0.0000000E+00	2998.890
IMP_REL_ARC(13, 21)	0.0000000E+00	3332.100
IMP_REL_ARC(13, 22)	0.0000000E+00	3332.100
IMP_REL_ARC(13, 23)	0.0000000E+00	1999.260
IMP_REL_ARC(14, 24)	0.0000000E+00	999.6301
IMP_REL_ARC(14, 25)	0.0000000E+00	1999.260
IMP_REL_ARC(14, 26)	0.0000000E+00	2332.470
IMP_REL_ARC(14, 27)	0.0000000E+00	2998.890

IMP_REL_ARC(14, 28)	0.0000000E+00	3332.100
IMP_REL_ARC(14, 29)	0.0000000E+00	3332.100
IMP_REL_ARC(14, 30)	0.0000000E+00	999.6301
IMP_REL_ARC(15, 24)	0.0000000E+00	1999.260
IMP_REL_ARC(15, 25)	0.0000000E+00	999.6301
IMP_REL_ARC(15, 26)	0.0000000E+00	1999.260
IMP_REL_ARC(15, 27)	0.0000000E+00	2332.470
IMP_REL_ARC(15, 28)	0.0000000E+00	2998.890
IMP_REL_ARC(15, 29)	0.0000000E+00	3332.100
IMP_REL_ARC(15, 30)	0.0000000E+00	3332.100
IMP_REL_ARC(16, 32)	0.0000000E+00	1999.260
IMP_REL_ARC(16, 33)	0.0000000E+00	999.6301
IMP_REL_ARC(17, 31)	0.0000000E+00	1999.260
IMP_REL_ARC(17, 32)	0.0000000E+00	2332.470
IMP_REL_ARC(17, 33)	0.0000000E+00	2998.890
IMP_REL_ARC(17, 34)	0.0000000E+00	3332.100
IMP_REL_ARC(17, 37)	0.0000000E+00	3332.100
IMP_REL_ARC(18, 31)	0.0000000E+00	1999.260
IMP_REL_ARC(18, 34)	0.0000000E+00	999.6301
IMP_REL_ARC(18, 35)	0.0000000E+00	1999.260
IMP_REL_ARC(19, 39)	0.0000000E+00	2332.470
IMP_REL_ARC(20, 40)	0.0000000E+00	2998.890
IMP_REL_ARC(21, 36)	0.0000000E+00	3332.100
IMP_REL_ARC(22, 37)	0.0000000E+00	3332.100
IMP_REL_ARC(23, 38)	0.0000000E+00	1999.260
IMP_REL_ARC(24, 41)	0.0000000E+00	3332.100
IMP_REL_ARC(25, 41)	0.0000000E+00	3332.100
IMP_REL_ARC(26, 41)	0.0000000E+00	3332.100
IMP_REL_ARC(27, 41)	0.0000000E+00	3332.100
IMP_REL_ARC(28, 41)	0.0000000E+00	3332.100
IMP_REL_ARC(29, 41)	0.0000000E+00	3332.100
IMP_REL_ARC(30, 41)	0.0000000E+00	3332.100
IMP_REL_ARC(31, 41)	0.0000000E+00	3332.100
IMP_REL_ARC(32, 41)	0.0000000E+00	3332.100
IMP_REL_ARC(33, 41)	0.0000000E+00	3332.100
IMP_REL_ARC(34, 41)	0.0000000E+00	3332.100
IMP_REL_ARC(35, 41)	0.0000000E+00	3332.100
IMP_REL_ARC(36, 41)	0.0000000E+00	3332.100
IMP_REL_ARC(37, 41)	0.0000000E+00	3332.100
IMP_REL_ARC(38, 41)	0.0000000E+00	3332.100
IMP_REL_ARC(39, 41)	0.0000000E+00	3332.100
IMP_REL_ARC(40, 41)	0.0000000E+00	3332.100
FLOW(1, 2)	910.2857	0.0000000E+00
FLOW(1, 3)	1011.429	0.0000000E+00
FLOW(1, 4)	622.4506	0.0000000E+00
FLOW(1, 5)	606.8571	0.0000000E+00
FLOW(1, 6)	180.0000	0.0000000E+00
FLOW(1, 7)	576.0000	0.0000000E+00
FLOW(2, 12)	910.2857	0.0000000E+00
FLOW(3, 12)	1011.429	0.0000000E+00
FLOW(4, 8)	622.4506	0.0000000E+00
FLOW(5, 12)	606.8571	0.0000000E+00

FLOW(6, 9)	180.0000	0.0000000E+00
FLOW(7, 10)	576.0000	0.0000000E+00
FLOW(8, 11)	622.4506	0.0000000E+00
FLOW(9, 12)	180.0000	0.0000000E+00
FLOW(10, 12)	576.0000	0.0000000E+00
FLOW(11, 12)	622.4506	0.0000000E+00
FLOW(12, 13)	3077.522	0.0000000E+00
FLOW(12, 24)	31.50000	0.0000000E+00
FLOW(12, 25)	18.00000	0.0000000E+00
FLOW(12, 39)	360.0000	0.0000000E+00
FLOW(12, 40)	420.0000	0.0000000E+00
FLOW(13, 14)	0.0000000E+00	0.0000000E+00
FLOW(13, 15)	829.1209	0.0000000E+00
FLOW(13, 16)	252.0000	0.0000000E+00
FLOW(13, 17)	0.0000000E+00	0.0000000E+00
FLOW(13, 18)	0.0000000E+00	0.0000000E+00
FLOW(13, 19)	90.14244	0.0000000E+00
FLOW(13, 20)	105.1378	0.0000000E+00
FLOW(13, 21)	829.1209	0.0000000E+00
FLOW(13, 22)	720.0000	0.0000000E+00
FLOW(13, 23)	252.0000	0.0000000E+00
FLOW(14, 24)	0.0000000E+00	0.0000000E+00
FLOW(14, 25)	0.0000000E+00	0.0000000E+00
FLOW(14, 26)	0.0000000E+00	0.0000000E+00
FLOW(14, 27)	0.0000000E+00	0.0000000E+00
FLOW(14, 28)	0.0000000E+00	0.0000000E+00
FLOW(14, 29)	0.0000000E+00	0.0000000E+00
FLOW(14, 30)	0.0000000E+00	0.0000000E+00
FLOW(15, 24)	0.6175218E-03	0.0000000E+00
FLOW(15, 25)	0.0000000E+00	0.0000000E+00
FLOW(15, 26)	0.0000000E+00	0.0000000E+00
FLOW(15, 27)	0.0000000E+00	0.0000000E+00
FLOW(15, 28)	0.0000000E+00	0.0000000E+00
FLOW(15, 29)	0.0000000E+00	0.0000000E+00
FLOW(15, 30)	829.1203	0.0000000E+00
FLOW(16, 32)	0.0000000E+00	0.0000000E+00
FLOW(16, 33)	252.0000	0.0000000E+00
FLOW(17, 31)	0.0000000E+00	0.0000000E+00
FLOW(17, 32)	0.0000000E+00	0.0000000E+00
FLOW(17, 33)	0.0000000E+00	0.0000000E+00
FLOW(17, 34)	0.0000000E+00	0.0000000E+00
FLOW(17, 37)	0.0000000E+00	0.0000000E+00
FLOW(18, 31)	0.0000000E+00	0.0000000E+00
FLOW(18, 34)	0.0000000E+00	0.0000000E+00
FLOW(18, 35)	0.0000000E+00	0.0000000E+00
FLOW(19, 39)	90.14244	0.0000000E+00
FLOW(20, 40)	105.1378	0.0000000E+00
FLOW(21, 36)	829.1209	0.0000000E+00
FLOW(22, 37)	720.0000	0.0000000E+00
FLOW(23, 38)	252.0000	0.0000000E+00
FLOW(24, 41)	31.50062	0.0000000E+00
FLOW(25, 41)	18.00000	0.0000000E+00

FLOW(26, 41)	0.0000000E+00	0.0000000E+00
FLOW(27, 41)	0.0000000E+00	0.0000000E+00
FLOW(28, 41)	0.0000000E+00	0.0000000E+00
FLOW(29, 41)	0.0000000E+00	0.0000000E+00
FLOW(30, 41)	829.1203	0.0000000E+00
FLOW(31, 41)	0.0000000E+00	0.0000000E+00
FLOW(32, 41)	0.0000000E+00	0.0000000E+00
FLOW(33, 41)	252.0000	0.0000000E+00
FLOW(34, 41)	0.0000000E+00	0.0000000E+00
FLOW(35, 41)	0.0000000E+00	0.0000000E+00
FLOW(36, 41)	829.1209	0.0000000E+00
FLOW(37, 41)	720.0000	0.0000000E+00
FLOW(38, 41)	252.0000	0.0000000E+00
FLOW(39, 41)	450.1424	0.0000000E+00
FLOW(40, 41)	525.1378	0.0000000E+00
Y_ARC(1, 2)	0.0000000E+00	414.5605
Y_ARC(1, 3)	0.0000000E+00	414.5605
Y_ARC(1, 4)	0.0000000E+00	414.5605
Y_ARC(1, 5)	0.0000000E+00	414.5605
Y_ARC(1, 6)	0.0000000E+00	414.5605
Y_ARC(1, 7)	0.0000000E+00	414.5605
Y_ARC(2, 12)	0.8916414E-01	0.0000000E+00
Y_ARC(3, 12)	0.1802477	0.0000000E+00
Y_ARC(4, 8)	0.0000000E+00	414.5605
Y_ARC(5, 12)	0.0000000E+00	414.5605
Y_ARC(6, 9)	0.0000000E+00	414.5605
Y_ARC(7, 10)	0.0000000E+00	414.5605
Y_ARC(8, 11)	0.0000000E+00	414.5605
Y_ARC(9, 12)	0.0000000E+00	414.5605
Y_ARC(10, 12)	0.0000000E+00	414.5605
Y_ARC(11, 12)	0.0000000E+00	414.5605
Y_ARC(12, 13)	0.7305881	0.0000000E+00
Y_ARC(12, 24)	0.0000000E+00	414.5605
Y_ARC(12, 25)	0.0000000E+00	414.5605
Y_ARC(12, 39)	0.0000000E+00	414.5605
Y_ARC(12, 40)	0.0000000E+00	414.5605
Y_ARC(13, 14)	0.0000000E+00	414.5605
Y_ARC(13, 15)	0.0000000E+00	414.5605
Y_ARC(13, 16)	0.0000000E+00	414.5605
Y_ARC(13, 17)	0.0000000E+00	414.5605
Y_ARC(13, 18)	0.0000000E+00	414.5605
Y_ARC(13, 19)	0.0000000E+00	414.5605
Y_ARC(13, 20)	0.0000000E+00	414.5605
Y_ARC(13, 21)	0.0000000E+00	414.5605
Y_ARC(13, 22)	0.0000000E+00	414.5605
Y_ARC(13, 23)	0.0000000E+00	414.5605
Y_ARC(14, 24)	0.0000000E+00	414.5605
Y_ARC(14, 25)	0.0000000E+00	414.5605
Y_ARC(14, 26)	0.0000000E+00	414.5605
Y_ARC(14, 27)	0.0000000E+00	414.5605
Y_ARC(14, 28)	0.0000000E+00	414.5605
Y_ARC(14, 29)	0.0000000E+00	414.5605

Y_ARC(14, 30)	0.0000000E+00	414.5605
Y_ARC(15, 24)	0.0000000E+00	414.5605
Y_ARC(15, 25)	0.0000000E+00	414.5605
Y_ARC(15, 26)	0.0000000E+00	414.5605
Y_ARC(15, 27)	0.0000000E+00	414.5605
Y_ARC(15, 28)	0.0000000E+00	414.5605
Y_ARC(15, 29)	0.0000000E+00	414.5605
Y_ARC(15, 30)	0.0000000E+00	414.5605
Y_ARC(16, 32)	0.0000000E+00	414.5605
Y_ARC(16, 33)	0.0000000E+00	414.5605
Y_ARC(17, 31)	0.0000000E+00	414.5605
Y_ARC(17, 32)	0.0000000E+00	414.5605
Y_ARC(17, 33)	0.0000000E+00	414.5605
Y_ARC(17, 34)	0.0000000E+00	414.5605
Y_ARC(17, 37)	0.0000000E+00	414.5605
Y_ARC(18, 31)	0.0000000E+00	414.5605
Y_ARC(18, 34)	0.0000000E+00	414.5605
Y_ARC(18, 35)	0.0000000E+00	414.5605
Y_ARC(19, 39)	0.0000000E+00	414.5605
Y_ARC(20, 40)	0.0000000E+00	414.5605
Y_ARC(21, 36)	0.0000000E+00	414.5605
Y_ARC(22, 37)	0.0000000E+00	414.5605
Y_ARC(23, 38)	0.0000000E+00	414.5605
Y_ARC(24, 41)	0.0000000E+00	414.5605
Y_ARC(25, 41)	0.0000000E+00	414.5605
Y_ARC(26, 41)	0.0000000E+00	414.5605
Y_ARC(27, 41)	0.0000000E+00	414.5605
Y_ARC(28, 41)	0.0000000E+00	414.5605
Y_ARC(29, 41)	0.0000000E+00	414.5605
Y_ARC(30, 41)	0.0000000E+00	414.5605
Y_ARC(31, 41)	0.0000000E+00	414.5605
Y_ARC(32, 41)	0.0000000E+00	414.5605
Y_ARC(33, 41)	0.0000000E+00	414.5605
Y_ARC(34, 41)	0.0000000E+00	414.5605
Y_ARC(35, 41)	0.0000000E+00	414.5605
Y_ARC(36, 41)	0.0000000E+00	414.5605
Y_ARC(37, 41)	0.0000000E+00	414.5605
Y_ARC(38, 41)	0.0000000E+00	414.5605
Y_ARC(39, 41)	0.0000000E+00	414.5605
Y_ARC(40, 41)	0.0000000E+00	414.5605

Row	Slack or Surplus	Dual Price
1	3077.901	1.000000
2	0.0000000E+00	0.0000000E+00
3	0.7000000	0.0000000E+00
4	0.3000000	0.0000000E+00
5	0.5000000	0.0000000E+00
6	0.2000000	0.0000000E+00
7	0.0000000E+00	0.0000000E+00
8	0.7000000	0.0000000E+00
9	0.3000000	0.0000000E+00
10	0.5000000	0.0000000E+00

11	0.2000000	0.0000000E+00
12	0.0000000E+00	0.0000000E+00
13	0.1571429	0.0000000E+00
14	0.3000000	0.0000000E+00
15	0.5000000	0.0000000E+00
16	0.2000000	0.0000000E+00
17	0.0000000E+00	0.0000000E+00
18	0.7000000	0.0000000E+00
19	0.3000000	0.0000000E+00
20	0.5000000	0.0000000E+00
21	0.5000000	0.0000000E+00
22	0.0000000E+00	0.0000000E+00
23	0.7000000	0.0000000E+00
24	0.3000000	0.0000000E+00
25	0.3000000	0.0000000E+00
26	0.2000000	0.0000000E+00
27	0.0000000E+00	0.0000000E+00
28	0.7000000	0.0000000E+00
29	0.3000000	0.0000000E+00
30	0.5000000	0.0000000E+00
31	0.2000000	0.0000000E+00
32	0.0000000E+00	0.0000000E+00
33	0.7000000	0.0000000E+00
34	0.3000000	0.0000000E+00
35	0.5000000	0.0000000E+00
36	0.2000000	0.0000000E+00
37	0.0000000E+00	0.0000000E+00
38	0.7000000	0.0000000E+00
39	0.3000000	0.0000000E+00
40	0.5000000	0.0000000E+00
41	0.5000000	0.0000000E+00
42	0.0000000E+00	0.0000000E+00
43	0.0000000E+00	0.0000000E+00
44	0.0000000E+00	0.0000000E+00
45	0.0000000E+00	0.0000000E+00
46	0.0000000E+00	0.0000000E+00
47	0.0000000E+00	0.0000000E+00
48	0.0000000E+00	0.0000000E+00
49	0.1000000	0.0000000E+00
50	0.0000000E+00	0.0000000E+00
51	0.0000000E+00	0.0000000E+00
52	0.4000000	0.0000000E+00
53	0.7000000	0.0000000E+00
54	0.4000000	0.0000000E+00
55	0.0000000E+00	0.0000000E+00
56	0.1000000	0.0000000E+00
57	0.0000000E+00	0.0000000E+00
58	0.3000000	0.0000000E+00
59	0.1000000	0.0000000E+00
60	0.4000000	0.0000000E+00
61	0.7000000	0.0000000E+00
62	0.4000000	0.0000000E+00

63	0.3000000	0.0000000E+00
64	0.0000000E+00	0.0000000E+00
65	0.0000000E+00	0.0000000E+00
66	0.4000000	0.0000000E+00
67	0.7000000	0.0000000E+00
68	0.4000000	0.0000000E+00
69	0.3000000	0.0000000E+00
70	0.1000000	0.0000000E+00
71	0.0000000E+00	0.0000000E+00
72	0.0000000E+00	0.0000000E+00
73	0.4000000	0.0000000E+00
74	0.7000000	0.0000000E+00
75	0.4000000	0.0000000E+00
76	0.3000000	0.0000000E+00
77	0.1000000	0.0000000E+00
78	0.0000000E+00	0.0000000E+00
79	0.0000000E+00	0.0000000E+00
80	0.7000000	0.0000000E+00
81	0.4000000	0.0000000E+00
82	0.7000000	0.0000000E+00
83	0.4000000	0.0000000E+00
84	0.3000000	0.0000000E+00
85	0.1000000	0.0000000E+00
86	0.0000000E+00	0.0000000E+00
87	0.0000000E+00	0.0000000E+00
88	0.4000000	0.0000000E+00
89	0.7000000	0.0000000E+00
90	0.4000000	0.0000000E+00
91	0.3000000	0.0000000E+00
92	0.1000000	0.0000000E+00
93	0.0000000E+00	0.0000000E+00
94	0.0000000E+00	0.0000000E+00
95	0.4000000	0.0000000E+00
96	0.7000000	0.0000000E+00
97	0.4000000	0.0000000E+00
98	0.3000000	0.0000000E+00
99	0.1000000	0.0000000E+00
100	0.0000000E+00	0.0000000E+00
101	0.0000000E+00	0.0000000E+00
102	0.4000000	0.0000000E+00
103	0.0000000E+00	0.0000000E+00
104	0.0000000E+00	0.0000000E+00
105	0.0000000E+00	0.0000000E+00
106	0.0000000E+00	0.0000000E+00
107	0.0000000E+00	0.0000000E+00
108	0.0000000E+00	0.0000000E+00
109	0.0000000E+00	0.0000000E+00
110	0.0000000E+00	0.0000000E+00
111	0.0000000E+00	0.0000000E+00
112	0.0000000E+00	0.0000000E+00
113	0.0000000E+00	0.0000000E+00
114	0.0000000E+00	0.0000000E+00

115	0.0000000E+00	0.0000000E+00
116	0.0000000E+00	0.0000000E+00
117	0.0000000E+00	0.0000000E+00
118	0.0000000E+00	0.0000000E+00
119	0.0000000E+00	0.0000000E+00
120	0.0000000E+00	0.5488976
121	0.0000000E+00	0.6277117
122	0.0000000E+00	0.3338744
123	0.0000000E+00	0.9637086
124	0.0000000E+00	0.9637086
125	0.0000000E+00	0.9637086
126	0.0000000E+00	0.2337121
127	730.2857	0.0000000E+00
128	435.4286	0.0000000E+00
129	0.0000000E+00	0.3961222
130	2970.478	0.0000000E+00
131	0.0000000E+00	0.3629138E-01
132	0.0000000E+00	0.3629138E-01
133	0.0000000E+00	0.3629138E-01
134	0.0000000E+00	0.3629138E-01
135	2400.000	0.0000000E+00
136	3010.879	0.0000000E+00
137	2628.000	0.0000000E+00
138	432.0000	0.0000000E+00
139	2016.000	0.0000000E+00
140	749.8576	0.0000000E+00
141	2054.862	0.0000000E+00
142	3970.879	0.0000000E+00
143	720.0000	0.0000000E+00
144	756.0000	0.0000000E+00
145	1008.000	0.0000000E+00
146	2304.000	0.0000000E+00
147	1680.000	0.0000000E+00
148	324.0000	0.0000000E+00
149	840.0000	0.0000000E+00
150	600.0000	0.0000000E+00
151	288.0000	0.0000000E+00
152	2015.999	0.0000000E+00
153	1152.000	0.0000000E+00
154	1440.000	0.0000000E+00
155	252.0000	0.0000000E+00
156	756.0000	0.0000000E+00
157	600.0000	0.0000000E+00
158	130.8797	0.0000000E+00
159	432.0000	0.0000000E+00
160	0.0000000E+00	0.0000000E+00
161	180.0000	0.0000000E+00
162	504.0000	0.0000000E+00
163	756.0000	0.0000000E+00
164	150.0000	0.0000000E+00
165	720.0000	0.0000000E+00
166	720.0000	0.0000000E+00

167	45.00000	0.0000000E+00
168	144.0000	0.0000000E+00
169	329.8576	0.0000000E+00
170	434.8622	0.0000000E+00
171	1570.879	0.0000000E+00
172	0.0000000E+00	0.0000000E+00
173	0.0000000E+00	0.0000000E+00
174	0.0000000E+00	0.4554179
175	0.0000000E+00	0.4098761
176	206.6703	0.0000000E+00
177	222.2637	0.0000000E+00
178	649.1209	0.0000000E+00
179	253.1209	0.0000000E+00
180	206.6703	0.0000000E+00
181	649.1209	0.0000000E+00
182	253.1209	0.0000000E+00
183	206.6703	0.0000000E+00
184	-0.8620398E-04	0.1347059
185	797.6209	0.0000000E+00
186	811.1209	0.0000000E+00
187	469.1209	0.0000000E+00
188	409.1209	0.0000000E+00
189	829.1209	0.0000000E+00
190	0.0000000E+00	0.0000000E+00
191	577.1209	0.0000000E+00
192	829.1209	0.0000000E+00
193	829.1209	0.0000000E+00
194	738.9784	0.0000000E+00
195	723.9830	0.0000000E+00
196	0.0000000E+00	0.0000000E+00
197	109.1209	0.0000000E+00
198	577.1209	0.0000000E+00
199	829.1209	0.0000000E+00
200	829.1209	0.0000000E+00
201	829.1209	0.0000000E+00
202	829.1209	0.0000000E+00
203	829.1209	0.0000000E+00
204	829.1209	0.0000000E+00
205	829.1209	0.0000000E+00
206	829.1203	0.0000000E+00
207	829.1209	0.0000000E+00
208	829.1209	0.0000000E+00
209	829.1209	0.0000000E+00
210	829.1209	0.0000000E+00
211	829.1209	0.0000000E+00
212	0.6175218E-03	0.0000000E+00
213	829.1209	0.0000000E+00
214	577.1209	0.0000000E+00
215	829.1209	0.0000000E+00
216	829.1209	0.0000000E+00
217	829.1209	0.0000000E+00
218	829.1209	0.0000000E+00

219	829.1209	0.0000000E+00
220	829.1209	0.0000000E+00
221	829.1209	0.0000000E+00
222	829.1209	0.0000000E+00
223	738.9784	0.0000000E+00
224	723.9830	0.0000000E+00
225	0.0000000E+00	0.0000000E+00
226	109.1209	0.0000000E+00
227	577.1209	0.0000000E+00
228	0.0000000E+00	0.7790403E-03
229	0.0000000E+00	3332.100
230	0.0000000E+00	0.7790403E-03
231	0.0000000E+00	3332.100
232	0.0000000E+00	414.5605
233	0.0000000E+00	-1.000000
234	0.0000000E+00	-1.000000
235	0.0000000E+00	-1.000000
236	0.0000000E+00	-1.000000
237	0.0000000E+00	-1.000000
238	0.0000000E+00	-1.000000
239	0.0000000E+00	-1.000000
240	0.0000000E+00	-0.6661256
241	0.0000000E+00	-0.3629138E-01
242	0.0000000E+00	-0.3629138E-01
243	0.0000000E+00	-0.4324135
244	0.0000000E+00	-0.3629138E-01
245	0.0000000E+00	0.0000000E+00
246	0.0000000E+00	0.0000000E+00
247	0.0000000E+00	0.0000000E+00
248	0.0000000E+00	0.0000000E+00
249	0.0000000E+00	0.0000000E+00
250	0.0000000E+00	0.0000000E+00
251	0.0000000E+00	0.0000000E+00
252	0.0000000E+00	0.0000000E+00
253	0.0000000E+00	0.0000000E+00
254	0.0000000E+00	0.0000000E+00
255	0.0000000E+00	0.0000000E+00
256	0.0000000E+00	0.0000000E+00
257	0.0000000E+00	0.0000000E+00
258	0.0000000E+00	0.0000000E+00
259	0.0000000E+00	0.0000000E+00
260	0.0000000E+00	0.0000000E+00
261	0.0000000E+00	0.0000000E+00
262	0.0000000E+00	0.0000000E+00
263	0.0000000E+00	0.0000000E+00
264	0.0000000E+00	0.0000000E+00
265	0.0000000E+00	0.0000000E+00
266	0.0000000E+00	0.0000000E+00
267	0.0000000E+00	0.0000000E+00
268	0.0000000E+00	0.0000000E+00
269	0.0000000E+00	0.0000000E+00
270	0.0000000E+00	0.0000000E+00

271	0.0000000E+00	0.0000000E+00
272	0.0000000E+00	0.0000000E+00
273	0.0000000E+00	0.0000000E+00

```
MODEL:

! Model D for Network A;

! Since LINGO numbers from 1 rather than 0, subtract 1 from variables to
translate back to diagram.  For example, arc (1,2) in LINGO program =
arc
(s,1) actual.;

SETS:
      NODE / 1..17 / : rel_node, imp_rel_node;
      ARC(NODE, NODE) / 1,2 1,3 1,4 1,5 2,13 2,14 2,15 3,6 3,15 4,10
4,12 5,15
            6,11 6,12 7,15 8,15 9,15 10,16 11,16 12,16 13,16 14,16 15,17
            16,7 16,8 16,9 / :
      cap, rel_arc, imp_cap, imp_rel_arc, flow, y;
ENDSETS

DATA:

      rel_node = 1 1 .3 .7 .5 .8 1 .3 .7 .5 .8 1 .3 .7 .5 .8 1;
      rel_arc = 1 1 1 1 1 1 .7 .3 .6 1 1 1 .6 .7 .6 .6 .3 1 .6 1 .7 1 1
.3 .6 .7;
      cap = 0 0 0 0
            1200 1200 1200 1200 1200 1200 1200 1200 1200 1200 4800 4800
4800 4800 4800
            4800 4800 4800 0 4800 4800 4800;
ENDDATA

MAX = Vef;

Vrel >= 0.98;

@FOR(NODE(I) : rel_node(I) + imp_rel_node(I) <= 1);

@FOR(ARC(I,J) : rel_arc(I,J) + imp_rel_arc(I,J) <= 1);

@FOR(ARC(I,J) | I #NE# 1 #AND# J #NE# 17 :
      flow(I,J) - ((cap(I,J) + imp_cap(I,J))*(rel_arc(I,J) +
imp_rel_arc(I,J))
      *(rel_node(J) + imp_rel_node(J))) <= 0);


@FOR(ARC(I,J) | I #NE# 1 #AND# J #NE# 17 :
      Vef - V + flow(I,J)*(1 - y(I,J)) <= 0);

@SUM(ARC(I,J) | I #NE# 1 #AND# J #NE# 17 :
      cap(I,J)*imp_cap(I,J)) <= Bcap;

@SUM(ARC(I,J) : rel_arc(I,J)*imp_rel_arc(I,J))
      + @SUM(NODE(K):rel_node(K)*imp_rel_node(K)) <= Brel;
```

```
0.0002*Bcap + 10000*Brel <= 2500;

@SUM(ARC(I,J) : y(I,J)) <= 1;

@FOR(NODE(J) | J #EQ# 1 :
      @SUM(ARC(J,K):flow(J,K)) - V = 0);

@FOR(NODE(J) | J #GT# 1 #AND# J #LT# 17 :
      @SUM(ARC(J,K):flow(J,K)) - @SUM(ARC(I,J):flow(I,J)) = 0);

@FOR(NODE(J) | J #EQ# 17 :
      V - @SUM(ARC(I,J):flow(I,J)) = 0);

! Rst and Rij terms.;

Rst = 1 - (1-(.5+imp_rel_node(5))*(.5+imp_rel_node(15)))
*(1-(.3+imp_rel_node(3))*(.6+imp_rel_arc(3,15))*(.5+imp_rel_node(15)))
*(1-
(.3+imp_rel_node(3))*(.6+imp_rel_arc(3,6))*(.8+imp_rel_node(6))*(.6+imp_
rel_arc(6,11))
      *(.8+imp_rel_node(11))*(.6+imp_rel_arc(11,16))*(.8+imp_rel_node(16
))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-
(.3+imp_rel_node(3))*(.6+imp_rel_arc(3,6))*(.8+imp_rel_node(6))*(.7+imp_
rel_arc(6,12))
      *(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.7+imp_rel_node(4))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.7+imp_rel_node(4))*(.5+imp_rel_node(10))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
```

```
*(1-(.3+imp_rel_node(13))*(.7+imp_rel_arc(13,16))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.7+imp_rel_node(14))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.5+imp_rel_node(15)));

R_3_6 = 1 - (1-(.5+imp_rel_node(5))*(.5+imp_rel_node(15)))
*(1-(.3+imp_rel_node(3))*(.6+imp_rel_arc(3,15))*(.5+imp_rel_node(15)))
*(1-(.7+imp_rel_node(4))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.7+imp_rel_node(4))*(.5+imp_rel_node(10))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.3+imp_rel_node(13))*(.7+imp_rel_arc(13,16))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.7+imp_rel_node(14))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.5+imp_rel_node(15)));

R_3_15 = 1 - (1-(.5+imp_rel_node(5))*(.5+imp_rel_node(15)))
*(1-
(.3+imp_rel_node(3))*(.6+imp_rel_arc(3,6))*(.8+imp_rel_node(6))*(.6+imp_
rel_arc(6,11))
```

```
      *(.8+imp_rel_node(11))*(.6+imp_rel_arc(11,16))*(.8+imp_rel_node(16
))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-
(.3+imp_rel_node(3))*(.6+imp_rel_arc(3,6))*(.8+imp_rel_node(6))*(.7+imp_
rel_arc(6,12))
      *(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.7+imp_rel_node(4))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.7+imp_rel_node(4))*(.5+imp_rel_node(10))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.3+imp_rel_node(13))*(.7+imp_rel_arc(13,16))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.7+imp_rel_node(14))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.5+imp_rel_node(15)));

R_6_11 = 1 - (1-(.5+imp_rel_node(5))*(.5+imp_rel_node(15)))
*(1-(.3+imp_rel_node(3))*(.6+imp_rel_arc(3,15))*(.5+imp_rel_node(15)))
```

```
*(1-
(.3+imp_rel_node(3))*(.6+imp_rel_arc(3,6))*(.8+imp_rel_node(6))*(.7+imp_
rel_arc(6,12))
      *(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.7+imp_rel_node(4))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.7+imp_rel_node(4))*(.5+imp_rel_node(10))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.3+imp_rel_node(13))*(.7+imp_rel_arc(13,16))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.7+imp_rel_node(14))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.5+imp_rel_node(15)));

R_6_12 = 1 - (1-(.5+imp_rel_node(5))*(.5+imp_rel_node(15)))
*(1-(.3+imp_rel_node(3))*(.6+imp_rel_arc(3,15))*(.5+imp_rel_node(15)))
*(1-
(.3+imp_rel_node(3))*(.6+imp_rel_arc(3,6))*(.8+imp_rel_node(6))*(.6+imp_
rel_arc(6,11))
      *(.8+imp_rel_node(11))*(.6+imp_rel_arc(11,16))*(.8+imp_rel_node(16
))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
```

```
      *(.5+imp_rel_node(15)))
*(1-(.7+imp_rel_node(4))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.7+imp_rel_node(4))*(.5+imp_rel_node(10))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.3+imp_rel_node(13))*(.7+imp_rel_arc(13,16))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.7+imp_rel_node(14))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.5+imp_rel_node(15)));

R_7_15 = 1 - (1-(.5+imp_rel_node(5))*(.5+imp_rel_node(15)))
*(1-(.3+imp_rel_node(3))*(.6+imp_rel_arc(3,15))*(.5+imp_rel_node(15)))
*(1-
(.3+imp_rel_node(3))*(.6+imp_rel_arc(3,6))*(.8+imp_rel_node(6))*(.6+imp_
rel_arc(6,11))
      *(.8+imp_rel_node(11))*(.6+imp_rel_arc(11,16))*(.8+imp_rel_node(16
))
      *(1-(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-
(.3+imp_rel_node(3))*(.6+imp_rel_arc(3,6))*(.8+imp_rel_node(6))*(.7+imp_
rel_arc(6,12))
      *(.8+imp_rel_node(16))
      *(1-(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
```

```
*(1-(.7+imp_rel_node(4))*(.8+imp_rel_node(16))
      *(1-(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.7+imp_rel_node(4))*(.5+imp_rel_node(10))*(.8+imp_rel_node(16))
      *(1-(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.3+imp_rel_node(13))*(.7+imp_rel_arc(13,16))*(.8+imp_rel_node(16))
      *(1-(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.7+imp_rel_node(14))*(.8+imp_rel_node(16))
      *(1-(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.5+imp_rel_node(15)));

R_8_15 = 1 - (1-(.5+imp_rel_node(5))*(.5+imp_rel_node(15)))
*(1-(.3+imp_rel_node(3))*(.6+imp_rel_arc(3,15))*(.5+imp_rel_node(15)))
*(1-
(.3+imp_rel_node(3))*(.6+imp_rel_arc(3,6))*(.8+imp_rel_node(6))*(.6+imp_
rel_arc(6,11))
      *(.8+imp_rel_node(11))*(.6+imp_rel_arc(11,16))*(.8+imp_rel_node(16
))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-
(.3+imp_rel_node(3))*(.6+imp_rel_arc(3,6))*(.8+imp_rel_node(6))*(.7+imp_
rel_arc(6,12))
      *(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.7+imp_rel_node(4))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.7+imp_rel_node(4))*(.5+imp_rel_node(10))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
```

```
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.3+imp_rel_node(13))*(.7+imp_rel_arc(13,16))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.7+imp_rel_node(14))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.5+imp_rel_node(15)));

R_9_15 = 1 - (1-(.5+imp_rel_node(5))*(.5+imp_rel_node(15)))
*(1-(.3+imp_rel_node(3))*(.6+imp_rel_arc(3,15))*(.5+imp_rel_node(15)))
*(1-
(.3+imp_rel_node(3))*(.6+imp_rel_arc(3,6))*(.8+imp_rel_node(6))*(.6+imp_
rel_arc(6,11))
      *(.8+imp_rel_node(11))*(.6+imp_rel_arc(11,16))*(.8+imp_rel_node(16
))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15))))
      *(.5+imp_rel_node(15)))
*(1-
(.3+imp_rel_node(3))*(.6+imp_rel_arc(3,6))*(.8+imp_rel_node(6))*(.7+imp_
rel_arc(6,12))
      *(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15))))
      *(.5+imp_rel_node(15)))
*(1-(.7+imp_rel_node(4))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15))))
      *(.5+imp_rel_node(15)))
*(1-(.7+imp_rel_node(4))*(.5+imp_rel_node(10))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15))))
      *(.5+imp_rel_node(15)))
*(1-(.3+imp_rel_node(13))*(.7+imp_rel_arc(13,16))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15))))
      *(.5+imp_rel_node(15)))
*(1-(.7+imp_rel_node(14))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
```

```
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15))))
      *(.5+imp_rel_node(15)))
*(1-(.5+imp_rel_node(15)));

R_11_16 = 1 - (1-(.5+imp_rel_node(5))*(.5+imp_rel_node(15)))
*(1-(.3+imp_rel_node(3))*(.6+imp_rel_arc(3,15))*(.5+imp_rel_node(15)))
*(1-
(.3+imp_rel_node(3))*(.6+imp_rel_arc(3,6))*(.8+imp_rel_node(6))*(.7+imp_
rel_arc(6,12))
      *(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.7+imp_rel_node(4))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.7+imp_rel_node(4))*(.5+imp_rel_node(10))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.3+imp_rel_node(13))*(.7+imp_rel_arc(13,16))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.7+imp_rel_node(14))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.5+imp_rel_node(15)));

R_13_16 = 1 - (1-(.5+imp_rel_node(5))*(.5+imp_rel_node(15)))
*(1-(.3+imp_rel_node(3))*(.6+imp_rel_arc(3,15))*(.5+imp_rel_node(15)))
*(1-
(.3+imp_rel_node(3))*(.6+imp_rel_arc(3,6))*(.8+imp_rel_node(6))*(.6+imp_
rel_arc(6,11))
```

```
      *(.8+imp_rel_node(11))*(.6+imp_rel_arc(11,16))*(.8+imp_rel_node(16
))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-
(.3+imp_rel_node(3))*(.6+imp_rel_arc(3,6))*(.8+imp_rel_node(6))*(.7+imp_
rel_arc(6,12))
      *(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.7+imp_rel_node(4))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.7+imp_rel_node(4))*(.5+imp_rel_node(10))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.7+imp_rel_node(14))*(.8+imp_rel_node(16))
      *(1-(1-(.3+imp_rel_arc(16,7))*(.6+imp_rel_arc(7,15)))
      *(1-
(.6+imp_rel_arc(16,8))*(.3+imp_rel_node(8))*(.6+imp_rel_arc(8,15)))
      *(1-
(.7+imp_rel_arc(16,9))*(.7+imp_rel_node(9))*(.3+imp_rel_arc(9,15))))
      *(.5+imp_rel_node(15)))
*(1-(.5+imp_rel_node(15)));

R_16_7 = R_7_15;

R_16_8 = R_8_15;

R_16_9 = R_9_15;

Vrel <= (R_3_6 - Rst + 1)*(1-y(3,6)) + y(3,6);

Vrel <= (R_3_15 - Rst + 1)*(1-y(3,15)) + y(3,15);

Vrel <= (R_6_11 - Rst + 1)*(1-y(6,11)) + y(6,11);
```

```
Vrel <= (R_6_12 - Rst + 1)*(1-y(6,12)) + y(6,12);

Vrel <= (R_7_15 - Rst + 1)*(1-y(7,15)) + y(7,15);

Vrel <= (R_8_15 - Rst + 1)*(1-y(8,15)) + y(8,15);

Vrel <= (R_9_15  - Rst + 1)*(1-y(9,15)) + y(9,15);

Vrel <= (R_11_16 - Rst + 1)*(1-y(11,16)) + y(11,16);

Vrel <= (R_13_16 - Rst + 1)*(1-y(13,16)) + y(13,16);

Vrel <= (R_16_7 - Rst + 1)*(1-y(16,7)) + y(16,7);

Vrel <= (R_16_8 - Rst + 1)*(1-y(16,8)) + y(16,8);

Vrel <= (R_16_9 - Rst + 1)*(1-y(16,9)) + y(16,9);

END
```

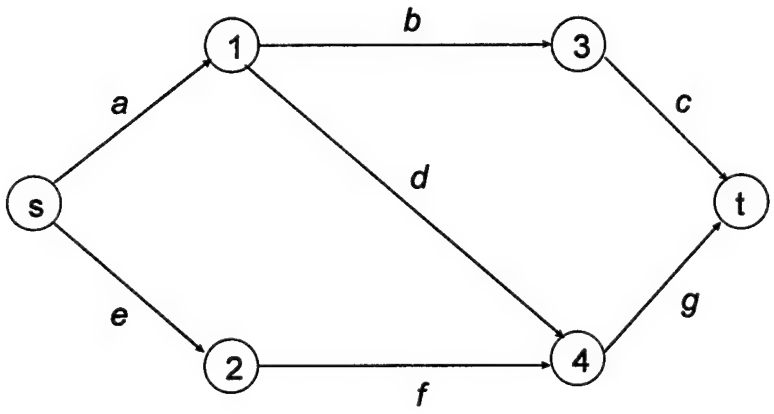
SUMMARIZED OUTPUT FOR NEWORK A: MODEL D

Network A						
efficient frontier analysis						
Vef	*Vrel*	*range equalized Vef*	*range equalized Vrel*	*l-1*	*l-2*	*l-inf*
5000	1	1.000	0.000	1.000	1.000	1.000
5500	0.996	0.857	0.174	1.031	0.875	0.857
6000	0.993	0.714	0.304	1.019	0.776	0.714
7500	0.987	0.286	0.565	0.851	0.633	0.565
8500	0.977	0.000	1.000	1.000	1.000	1.000

Appendix C: Manual and Computer Calculation of Reliability Expressions

The reader is referred to Shier's excellent monograph, *Network Reliability and Algebraic Structures* [27], for a complete and rigorous discussion on the calculation of R_{st}, two-terminal reliability for probabilistic networks.

Two techniques were employed to calculate *Rst*; namely, factoring described in Chapter II and Shier's algebraic approach. For example, assume the letters by the arcs in the sample network represent component reliabilities.



The reliability expression is the sum of the paths from source to sink, using Shier's algebra to implement the inclusion-exclusion formula for system reliability. In the case of the sample network:

$$
\begin{aligned}
&R_{st} = abc \oplus adg \oplus efg = \\
&\left[abc + adg - \left(abc \otimes adg\right)\right] \oplus efg = \\
&\left(abc + adg - abcdg\right) \oplus efg = \\
&abc + adg - abcdg + efg - \left[\left(abc + adg - abcdg\right) \otimes efg\right] = \\
&abc + adg + efg - abcdg - \left(abcefg + adefg - abcdefg\right) = \\
&abc + adg + efg - abcdg - adefg - abcefg + abcdefg
\end{aligned}
$$

The following pages contain a program in C++ which implements this approach. The programmer, Eric Moyer of Wright State University, is a colleague of the author, and his contribution to this thesis is much appreciated.

```
/This file is the final version of the reliability expression generator
//It reads a network from stdin and then writes out an analytical
expression
//for the source to sink reliability in terms of the reliabilities of
the
//individual edges.

#include "iostream.h"
#include "Network.h"
#include "PathExpression.h"

int main(){
        cout << "Reading in network." <<endl;
        Network network(cin);

        cout << "Converting network to circle plus expression."<<endl;
        PathExp exp(network.allPaths());

        cout << "Eliminating circle-plus operators." <<endl;
        exp.removeCirclePlus();

        cout << "Eliminating circle-times operators." <<endl;
        exp.removeCircleTimes();

        cout << "Eliminating minus signs." <<endl;
        exp.removeMinus();
        cout << "The final expression is: " <<endl;
        exp.printOnWithoutParentheses(cout);
        cout <<endl <<endl;

        return(0);
}
```

```
//Network objects store directed stochastic networks
//Nodes are lists of edges which start at that node
//Networks are arrays of nodes

//File format for reading in a network: text file of integers
//Blank lines are ignored.
//The file starts with the text "network1.0" on its own line then
//numNodes numEdges
//sourceNodeNumber sinkNodeNumber
//sourceNode destinationNode edgeId
//sourceNode destinationNode edgeId
//... (repeat until have entered numEdges edges on own line) ...
//ignores rest of file


//Billing
//25 Jan 1998 --        120 Min
//27 Jan 1998 --         20 Min
//30 Jan 1998 --        Start: 2:40 (-20 min) (-15 min)

//Lito -- 454-5891

#ifndef EM_NETWORK_H
#define EM_NETWORK_H

#include <iostream.h>
#include <list>
#include <vector>

//Index of a node in the array [0..largest in array]
typedef unsigned long NodeNum;

//Unique (within a given network) identifier of an edge [0..2^32-1]
//Has operator< and operator= and operator== and operator<<
typedef unsigned long EdgeId;
typedef EdgeId EdgeID;


struct Edge{
        Edge():id(0), destination(0){}
        Edge(EdgeId iid, NodeNum idestination):
                id(iid), destination(idestination){}

        Edge(Edge const & e): id(e.id), destination(e.destination){}

        EdgeId id; //Assumed to be unique

        //Source of the edge is the node in which the edge is listed

        //Destination is number of Node to which this edge goes.
        NodeNum destination;
```

```
        bool operator==(Edge const&b) const {return id==b.id;}
        bool operator<(Edge const&b)  const {return id< b.id;}
};

inline ostream & operator<<(ostream & out, Edge const & edge){
        out << edge.destination << " " << edge.id;

        return(out);
}


class Node:public list<Edge>{
public:
        //Create an unmarked node with no edges
        Node():list<Edge>(), m_isMarked(false){};
        Node(Node const &orig):
                list<Edge>(orig),
                m_isMarked(orig.isMarked()){};

        //Add an edge named id going from this node to the
        //destination node
        void addEdgeIdGoingTo(unsigned long id, NodeNum destination);

        //Used in depth first enumeration of paths
        bool isMarked() const{return(m_isMarked);}
        void mark(){m_isMarked=true;}
        void unmark(){m_isMarked=false;}
private:
        bool m_isMarked;
};

//Useful for creating a list of all paths
typedef list<EdgeId> BarePath;
typedef list<BarePath> BarePathList;

ostream& operator<<(ostream& out, BarePath const & path);
ostream& operator<<(ostream& out, BarePathList const & pathList);


class Network:public vector<Node>{
public:
        //Create a network from the text stream given by in
        //see top for file format
        Network(istream&in);

        //Treating a source to sink path as a list of edges, creates
        //a list of all source to sink paths.
        BarePathList allPaths();

        //Return the indices of the source and sink nodes
        NodeNum sourceNum() const{return(m_sourceNum);}
        NodeNum sinkNum() const{return(m_sinkNum);}
```

```
        //Return a reference to the source and sink nodes themselves
        Node &sourceNode(){ return (*this)[sourceNum()]; }
        Node &sinkNode(){ return (*this)[sinkNum()]; }

        //Return the number of nodes and the number of edges
        unsigned long numNodes() const { return(this->size()); }
        unsigned long numEdges() const;
protected:
        //Set the indices of the source and sink node
        void sourceNum(NodeNum newVal){m_sourceNum=newVal;}
        void sinkNum(NodeNum newVal){m_sinkNum=newVal;}

        //For each path from this node to the sink which passes
        //through each node on it only once and does not pass
        //through any marked nodes, that path is prefixed with
        //the given prefix and added to the end of list.  No
        //guarantees are made as to the order in which the paths
        //are added.
        //Note:
        //      If startNode is the sink node, then prefix is
        //      added to the list by itself and the function
        //      returns.
        void addAllPathsFromNodeThroughUnmarkedNodesWithPrefix(
                Node& startNode,
                BarePathList& list,
                BarePath& prefix);

private:
        friend ostream& operator<<(ostream& out, Network const & net);

        //Prints the edges of node number nodeNum on out in format
        //defined at beginning of Network.h
        void printEdgesFromNodeOn(NodeNum nodeNum, ostream & out) const;

        NodeNum m_sourceNum, m_sinkNum;
};

//Write to text file in the file format given at the top
ostream& operator<<(ostream& out, Network const & net);

#endif
```

```
/This file contains the implementation of the functions declared in the
//header file: Network.h


#include <assert.h>
#include "Network.h"
#include <iostream.h>
#include <stdlib.h>

#define UNIMPLEMENTED false

//Add an edge named id going from this node to the
//destination node
void Node::addEdgeIdGoingTo(unsigned long id, NodeNum destination){
        push_back(Edge(id, destination));
}

ostream& operator<<(ostream& out, BarePath const & path){
        BarePath::const_iterator iter=path.begin();
        for(unsigned long i=0; i<path.size(); ++i, ++iter){
                out << *iter << " ";
        }
        return(out);
}

ostream& operator<<(ostream& out, BarePathList const & pathList){
        BarePathList::const_iterator iter=pathList.begin();
        unsigned long i=0;
        for(; i + 1 < pathList.size(); ++i, ++iter){
                out << "(" << *iter << ") + ";
        }
        if( i < pathList.size() ){
                out << "(" << *iter << ")";}
        return(out);
}

void printOffByOneInputMessage(ostream &out, unsigned long numNodes){
        out  << endl
             << "Remember node numbers start at 0 so the first illegal "
             << "value is: " << numNodes <<endl
             << "If you mistakenly numbered from zero, you can fix this
"
             << "by increasing" << endl
             << "the number of nodes by 1.  This will not affect
reliability "
             << "results." << endl
             << "You will just have a detached node. " << endl;
}

template class list<Edge>;

//Create a network from the text stream given by in
```

```
//see top of Network.h for file format
Network::Network(istream& in){
        unsigned long numNodes, numEdges;

        in >> numNodes >> numEdges;

        //resize(numNodes, Node())
        if(numNodes<size()){
                erase(begin() + numNodes, end());
        }else{
                insert(end(), numNodes - size(), Node());
        }
        cerr<<numNodes<<":" << size() << "/" << capacity() << endl;

        NodeNum sourceNumber, sinkNumber;
        in >> sourceNumber >> sinkNumber;
        sourceNum(sourceNumber); sinkNum(sinkNumber);

        if(sourceNumber>=numNodes || sinkNumber>=numNodes){
                cerr << "Error in file: source or sink number too big"
<< endl;
                printOffByOneInputMessage(cerr, numNodes); exit(-1);
        }

        for(unsigned long i=0; i<numEdges; ++i){
                NodeNum source, destination; EdgeId id;
                in >> source >> destination >> id;

                if(source >= numNodes){
                        cerr << "Error in file: edge going from too
large a node number"<<endl
                             << "Node number was: " << source <<endl;
                        printOffByOneInputMessage(cerr, numNodes);
exit(-1);
                }
                if(destination >= numNodes){
                        cerr << "Error in file: edge going to too large
a node number" << endl
                             << "Node number was: " << destination
<<endl;
                        printOffByOneInputMessage(cerr, numNodes);
exit(-1);
                }

                (*this)[source].addEdgeIdGoingTo(id, destination);
        }
//      cerr << "Debug - Exiting read network routine" << endl;
}

//Treating a source to sink path as a list of edges, creates
//a list of all source to sink paths.
BarePathList Network::allPaths(){
```

```
        BarePathList list;
        BarePath prefix;

        addAllPathsFromNodeThroughUnmarkedNodesWithPrefix(
                sourceNode(), list, prefix);

        return(list);
}

//Return the number of edges in this network
unsigned long Network::numEdges() const{
        unsigned long total=0;
        for(unsigned long i=0; i<numNodes(); ++i){
                //for each node, add number of edges from node to total
edges
                total+=(*this)[i].size();
        };
        return(total);
}

//For each path from this node to the sink which passes
//through each node on it only once and does not pass
//through any marked nodes, that path is prefixed with
//the given prefix and added to the end of list.  No
//guarantees are made as to the order in which the paths
//are added.
//Note:
// If startNode is the sink node, then prefix is
// added to the list by itself and the function
// returns.
void Network::addAllPathsFromNodeThroughUnmarkedNodesWithPrefix(
      Node& startNode,
      BarePathList& list,
      BarePath& prefix)
{
        if( &startNode == &(sinkNode()) ){
                list.push_back(prefix); return; }

        startNode.mark();

        Node::const_iterator iter=startNode.begin();
        for (unsigned long i=0; i<startNode.size(); ++i, ++iter ){
                Node &dest=(*this)[ (*iter).destination ];

                if( ! dest.isMarked() ){
                        prefix.push_back((*iter).id);

addAllPathsFromNodeThroughUnmarkedNodesWithPrefix(
                                dest, list, prefix);
                        prefix.pop_back();
                }
        }
```

```
        startNode.unmark();
}

//Prints the edges of node number nodeNum on out in format
//defined at beginning of Network.h
void Network::printEdgesFromNodeOn(NodeNum nodeNum, ostream & out)
const{
        Node const& node=(*this)[nodeNum];

        Node::const_iterator iter=node.begin();
        for (unsigned long i=0; i<node.size(); ++i, ++iter )
                out << nodeNum << " " << *iter << endl;
}


//Write to text file in the file format given at the top of Network.h
ostream& operator<<(ostream& out, Network const & net){

        out << "network1.0" << endl;

        unsigned long numNodes=net.numNodes();
        out << numNodes << " " << net.numEdges() << endl;

        out << net.sourceNum() << " " << net.sinkNum() << endl;

        for(unsigned long i=0; i<numNodes; ++i){
                net.printEdgesFromNodeOn(i, out);
                out << endl;
        }
        return(out);
}
```

```
#ifndef EM_PATH_EXPRESSION_H
#define EM_PATH_EXPRESSION_H

#include "Network.h"
#include <assert.h>
#include <iostream.h>
#include <algorithm>
#include <list>

#ifndef EM_NETWORK_H
//for debugging
typedef unsigned long EdgeID;
typedef list<EdgeList> BarePathList;
#endif
typedef list<EdgeID> EdgeList;

//Note: do a search and replace to fix these 18r.
//PTH=path, CP=CirclePlus, CT=CircleTimes, PL=Plus, MI=Minus
enum NodeTypeID {PTH=0, CP, CT, PL, MI} ;

//I could have done this with virtual functions and several classes.
However,
//I fought with the language so long when I needed to remove a node of
one type
//and replace it with a node of another type, that I just resorted to
this --
//which, despite its ugliness, seems to be cleaner.  What I wouldn't
give for a
//'become' statement like in Smalltalk.
class PathExpNode{
public:
        //Creates a new PathExpNode for an operator.
        //Does BAD THINGS if i_typeID is PTH
        PathExpNode(
                NodeTypeID i_typeID,
                PathExpNode *i_lhs,
                PathExpNode *i_rhs):m_typeID(i_typeID),
                                    m_isNegative(false),
                                    m_edgeList(),

m_lhs(i_lhs),

m_rhs(i_rhs){ assert(i_typeID != PTH); }

        //Creates a new PathExpNode for a path initialized to given
values.
        //Does BAD THINGS if i_typeID is anything but PTH
        PathExpNode(
                NodeTypeID i_typeID,
                bool i_isNegative,
                EdgeList const & i_edgeList):m_typeID(i_typeID),
                                             m_isNegative(i_isNegative),
```

```
m_edgeList(i_edgeList),

m_lhs(NULL),

m_rhs(NULL){ assert(i_typeID == PTH);}

        //Copy constructor.  Not only copies this node, but all its
subtrees as well
        PathExpNode(
                PathExpNode const & p): m_typeID( p.typeID() ),
                                m_isNegative(p.m_isNegative),
                                        m_edgeList(p.m_edgeList),
                                        m_lhs(NULL),
                                        m_rhs(NULL){
                if(typeID() == PTH) return; //No subtrees to copy.
                //Copy subtrees
                lhs(new PathExpNode( *(p.lhs()) ));
                rhs(new PathExpNode( *(p.rhs()) ));
        }

        //Deletes this node and all under it (assuming they were alloc'd
with new)
        ~PathExpNode(){ if(typeID()!=PTH){ delete lhs(); delete rhs(); }
}

        void printOn(ostream& out){
                switch (typeID()){
                        case PTH: PTH_printOn(out); break;

                        case CP: CP_printOn(out); break;

                        case CT: CT_printOn(out); break;

                        case PL: PL_printOn(out); break;

                        case MI: MI_printOn(out); break;

                        default:
                                cerr <<"Unknown
typeID:"<<typeID()<<endl; assert(0);
                };
        }

        //Prints this expression without most of the parentheses.
        //As yet, does not understand anything about operator precedence
        //or associativity.  Use at your own risk (when parentheses are
        //known not to be needed)
        void printOnWithoutParentheses(ostream& out);

        //Removes CircleTimes from this node and all its subtrees
        void removeCircleTimes(){
```

```
                switch (typeID()){
                        case PTH: return; break;

                        case CT: CT_removeCircleTimes(); break;

                        case CP: case PL: case MI:
                                lhs()->removeCircleTimes(); rhs()-
>removeCircleTimes();
                        break;

                        default:
                                cerr <<"Unknown
typeID:"<<typeID()<<endl; assert(0);
                };
        }

        //Removes CirclePlus from this node and all its subtrees
        void removeCirclePlus(){
                switch (typeID()){
                        case PTH: return; break;

                        case CP: CP_removeCirclePlus(); break;

                        case CT: case PL: case MI:
                                lhs()->removeCirclePlus(); rhs()-
>removeCirclePlus();
                        break;

                        default:
                                cerr <<"Unknown
typeID:"<<typeID()<<endl; assert(0);
                };
        }

        //Removes Minus from this node and all its subtrees
        void removeMinus();

        //Appends the path specified by edgeList to the expression at
this point,
        //joined by an operator of type operatorType.
        //BAD THINGS if operatorType==PTH
        void appendEdgeList(NodeTypeID operatorType, EdgeList const
&i_edgeList);


protected:
        //Removes Minus from this node and all its subtrees and negates
any
        //paths it comes across.  -- Support routine for remove minus.
        void removeMinusAndNegate();

        //Become path function
```

```
        //Note: DOES NOT DELETE subtrees.  Please take care of them
before calling
        //Note: does not always use the accessors since they are picky
about values.
        void becomePathNode(bool i_isNegative, EdgeList const&
i_edgeList);

        //Become operator function -- BAD THINGS happen if newType ==
PTH
        //Note: DOES NOT DELETE subtrees.  Please take care of them
before calling
        void becomeOperatorNode( NodeTypeID newType, PathExpNode*i_lhs,
                                 PathExpNode*i_rhs);


        //Removal functions for CirclePlus and CircleTimes
        void CP_removeCirclePlus();

        void CT_removeCircleTimes();
        //3 cases depending on whether the child nodes are paths
        void CT_removeCircleTimesPathPath();
        void CT_removeCircleTimesPathOper();
        void CT_removeCircleTimesOperOper();


////////////////////////////////////////////////////////////////////////
///
        //Print Methods -- these methods take the place of virtual print
function//

////////////////////////////////////////////////////////////////////////
///

        void PTH_printOn(ostream& out);
        void CP_printOn(ostream& out);
        void CT_printOn(ostream& out);
        void PL_printOn(ostream& out);
        void MI_printOn(ostream& out);


        void CP_printOnWithoutParentheses(ostream& out);
        void CT_printOnWithoutParentheses(ostream& out);
        void PL_printOnWithoutParentheses(ostream& out);
        void MI_printOnWithoutParentheses(ostream& out);


////////////////////////////////////////////////////////////////////////
/
        //Accessor Methods -- These do typechecking to make sure member
exists //
        //                      in this type of object
//
```

```
////////////////////////////////////////////////////////////////////////
/

        //Getter for the type.  Protected so that routines using this
class will
        //have to use it just as if it were a base class -- and they
won't be
        //allowed to cast.
        NodeTypeID typeID() const{ return(m_typeID); }

        //Be careful with this one.  To get this kind of power is the
entire reason
        //for bothering with this convoluted code.
        void typeID(NodeTypeID newTypeID){ m_typeID=newTypeID; }

        bool isNegative() const{ assert(typeID()==PTH); return
m_isNegative; }
        void isNegative(bool newVal){ assert(typeID()==PTH);
m_isNegative=newVal; }

        EdgeList const& edgeList() const{ assert(typeID()==PTH); return
m_edgeList; }
        EdgeList & edgeList() { assert(typeID()==PTH); return
m_edgeList;}
        void edgeList(EdgeList const& newVal){
                assert(typeID()==PTH); edgeList()=newVal; }

        PathExpNode *lhs() const{ assert(typeID()!=PTH); return m_lhs; }
        void lhs(PathExpNode *newVal) {
                assert(typeID()!=PTH && newVal!=NULL); m_lhs=newVal; }

        PathExpNode *rhs() const{ assert(typeID()!=PTH); return m_rhs; }
        void rhs(PathExpNode *newVal) {
                assert(typeID()!=PTH && newVal!=NULL); m_rhs=newVal; }

private:
        NodeTypeID m_typeID;

        //Members from Paths
        bool m_isNegative;   //True if this path is negated
        EdgeList m_edgeList; //Sorted list of edges in the path
                             //To save space, should be empty when not a
path.

        //Members from Operators
        //m_lhs and m_rhs are NULL iff this node is a path
        PathExpNode *m_lhs, *m_rhs; //Left hand and right hand nodes.

};
```

```
//Path expression -- mostly holds the root of the expression tree and
the
//knowledge of how to build it
class PathExp{
public:
        //Creates a PathExpression made by CirclePlussing all
        //the paths in orig.
        PathExp(BarePathList const&orig);

        //Simplifies the path expression down to the level of there
being no
        //CirclePlus operators.  Does this by (A (+) B) = (A + B) - (A
(*) B)
        void removeCirclePlus(){root() -> removeCirclePlus();}

        //Simplifies the path expression down to the level of there
being no
        //CircleTimes operators.  Does this by distributing
        //A (*) (B ?? C)=(A (*) B) ?? (A (*) C)
        //And by using the fact that if A and B are EdgeLists then
        //A (*) B  is just a new edge list, the contents of which are
the union of
        //the contents of A and B
        void removeCircleTimes(){root() -> removeCircleTimes();}

        //Removes all minus signs in the tree turning them into positive
or
        //negative coefficients on the paths.
        void removeMinus(){root() -> removeMinus();}


        //The printing function
        void printOn(ostream& out){ root()->printOn(out); }

        //The printing function without annoying parentheses
        //Use only when you know parentheses wont be needed
        void printOnWithoutParentheses(ostream& out){
                root()->printOnWithoutParentheses(out); }

        //Accessors for root
        PathExpNode *root() const{return m_root;}
        void root(PathExpNode *newRoot){
                assert(newRoot!=0 || "Assigning NULL to root of
PathExp"==0);
                m_root=newRoot;
        }

private:
        //Should never be NULL
        PathExpNode* m_root;

};
```

```
#endif
```

```
#include "PathExpression.h"

PathExp::PathExp(BarePathList const & list){
        BarePathList::const_iterator iter=list.begin();

        if(list.size()>0){
                EdgeList l(*iter);
                l.sort();
                m_root=new PathExpNode(PTH, false, l);
                ++iter;
        } else {
                EdgeList empty; m_root=new PathExpNode(PTH, false,
empty); }

        for(unsigned long i=1; i<list.size(); ++i, ++iter){
                EdgeList l(*iter);
                l.sort();
                root()->appendEdgeList(CP, l);
        }
}

//Given that this node is a CirclePlus, removes CirclePlusses from it
//and all its subtrees
void PathExpNode::CP_removeCirclePlus(){
        assert(typeID() == CP);

        lhs()->removeCirclePlus();

        PathExpNode *oldLHS=lhs(), *oldRHS=rhs();

        PathExpNode *newLHS= new PathExpNode(PL, oldLHS, oldRHS);
        PathExpNode *newRHS=
                new PathExpNode(CT, new PathExpNode(*oldLHS), new
PathExpNode(*oldRHS));

        becomeOperatorNode(MI, newLHS, newRHS);

        rhs()->removeCirclePlus();
}

//Given that this node is a CircleTimes, removes CircleTimeses from it
//and all its subtrees
void PathExpNode::CT_removeCircleTimes(){
        assert(typeID() == CT);

        //note: didn't test any situations in which this could be true.
        //however, this takes care of the case of an operator over which
        //circleTimes can't distribute -- since circleTimes is the only
such
        //operator.
        if(lhs()->typeID() == CT) lhs()->removeCircleTimes();
        if(rhs()->typeID() == CT) rhs()->removeCircleTimes();
```

```
        bool lhsPath = lhs()->typeID() == PTH;
        bool rhsPath = rhs()->typeID() == PTH;

        if(lhsPath)
                if(rhsPath){
                        CT_removeCircleTimesPathPath();
                        return;
                }else
                        CT_removeCircleTimesPathOper();
        else
                if(rhsPath){
                        //Swap two sides
                        PathExpNode* tmp=lhs();
                        lhs(rhs()); rhs(tmp);

                        CT_removeCircleTimesPathOper();
                }else
                        CT_removeCircleTimesOperOper();

        lhs()->removeCircleTimes();
        rhs()->removeCircleTimes();
}

template<class T>
bool is_sorted(list<T> const&l){
        list<T>::const_iterator iter=l.begin(), lastIter=iter;
        ++iter;
        for(unsigned long i=1; i<l.size(); ++i, ++iter)
                if(*iter<*lastIter) return(false);
        return(true);
}

//Given that this node is a CircleTimes
//And given that both children of this node are paths,
//Changes this node into a path whose edges are the union of the edges
in
//the children.
void PathExpNode::CT_removeCircleTimesPathPath(){

        assert(typeID() == CT);
        EdgeList const &lhsEdges=lhs()->edgeList();
        EdgeList const &rhsEdges=rhs()->edgeList();

        assert(is_sorted(lhsEdges));
        assert(is_sorted(rhsEdges));

        EdgeList newEdges;

        //Doesn't seem to work under old gnu compiler
        //set_union(lhsEdges.begin(), lhsEdges.end(),
        //          rhsEdges.begin(), rhsEdges.end(), newEdges.begin());
```

```
        newEdges=lhsEdges;
        EdgeList::const_iterator item=rhsEdges.begin();
        while(item != rhsEdges.end())
                newEdges.push_back(*item++);
        newEdges.sort();

        //Doing a unique
        EdgeList::iterator cur=newEdges.begin();
        EdgeList::iterator prev=cur;
        ++cur;
        while(cur != newEdges.end()){
                assert(prev != cur);
                if(*prev!=*cur){
                        prev=cur;
                        ++cur;
                }else{
                        EdgeList::iterator tmp=cur;
                        ++cur;
                        newEdges.erase(tmp);
                }
        }


        bool isNeg=lhs()->isNegative();
        if(rhs()->isNegative() ) isNeg = !isNeg;

        delete lhs(); delete rhs();
        becomePathNode(isNeg, newEdges);
}

//Given that this node is a CircleTimes
//And given that the left hand side is a path and the right hand side
//is an operator over which CircleTimes distributes, distributes the
//circleTimes over oper.
void PathExpNode::CT_removeCircleTimesPathOper(){
        assert(typeID() == CT);

        PathExpNode *path=lhs(),
                    *oldRHS=rhs();
        NodeTypeID   oldRHSOperator=rhs()->typeID();

        PathExpNode *newLHS=new PathExpNode(CT, path, oldRHS->lhs());

        oldRHS->becomeOperatorNode(CT, new PathExpNode(*path), oldRHS-
>rhs());

        becomeOperatorNode( oldRHSOperator, newLHS, oldRHS );
}

//Given that this node is a CircleTimes
//And given that both sides are operators and that the left hand side
//is an operator over which CircleTimes distributes, distributes the
```

```
//CircleTimes over oper.
void PathExpNode::CT_removeCircleTimesOperOper(){
        assert(typeID() == CT);

        //Simplifying expression of form
        //  (A ? B) (*) (C ?? D)
        //into
        //  (A (*) (C ?? D)) ? (B (*) (C ?? D))
        // This has the effect of moving the (*) down a level so it will
eventually
        // reach the bottom of the tree -- composed of paths.

        PathExpNode *a= lhs()->lhs(),
                    *b= lhs()->rhs(),
                    *cd= rhs(), //The expression (C ?? D)
                    *cdCopy= new PathExpNode (*cd);

        becomeOperatorNode(lhs()->typeID(), lhs(), rhs());

        lhs()->becomeOperatorNode(CT, a, cd);
        rhs()->becomeOperatorNode(CT, b, cdCopy);
}

//Removes all minuses in this tree and its subtrees
void PathExpNode::removeMinus(){
        switch(typeID()){
                case MI:
                        becomeOperatorNode(PL, lhs(), rhs());
                        lhs()->removeMinus();
                        rhs()->removeMinusAndNegate();
                break;

                case PTH: break;

                case CT: case CP: case PL:
                        lhs()->removeMinus();
                        rhs()->removeMinus();
                break;

                default:
                        cerr <<"Unknown typeID:"<<typeID()<<endl;
assert(0);
        }
        return;
}

//Removes all minuses in this tree and its subtrees while negating every
//path which it encounters.
void PathExpNode::removeMinusAndNegate(){
        switch(typeID()){
                case MI:
                        becomeOperatorNode(PL, lhs(), rhs());
```

```
                        lhs()->removeMinusAndNegate();
                        rhs()->removeMinus();
                break;

                case PTH: isNegative(!isNegative()); break;

                case CT: case CP: case PL:
                        lhs()->removeMinusAndNegate();
                        rhs()->removeMinusAndNegate();
                break;

                default:
                        cerr <<"Unknown typeID:"<<typeID()<<endl;
assert(0);
        }
        return;
}

//Appends the path specified by edgeList to the expression at this
point,
//joined by an operator of type operatorType.
//BAD THINGS if operatorType==PTH
void PathExpNode::appendEdgeList(NodeTypeID operatorType,
                                 EdgeList const &i_edgeList){
   assert(operatorType != PTH);

   PathExpNode *newNode=NULL;
   if(typeID() == PTH){
      newNode=new PathExpNode(typeID(), isNegative(), edgeList());
   }else{
      newNode=new PathExpNode(typeID(), lhs(), rhs());
   }

   PathExpNode *newPath=new PathExpNode(PTH, false, i_edgeList);

   becomeOperatorNode(operatorType, newNode, newPath);
}

////////////////////////////////////////////////////////////////////
//Become Methods -- allow the object to change its type (magic :)//
////////////////////////////////////////////////////////////////////
//Become path function
//Note: DOES NOT DELETE subtrees.  Please take care of them before
calling
//Note: does not always use the accessors since they are picky about
values.
void PathExpNode::becomePathNode(bool i_isNegative, EdgeList const&
i_edgeList){
   NodeTypeID originalType = typeID();

   if(originalType != PTH){ m_lhs=NULL; m_rhs=NULL; };
```

```
    typeID(PTH); isNegative(i_isNegative); edgeList()=i_edgeList;
}

//Become operator function -- BAD THINGS happen if newType == PTH
//Note: DOES NOT DELETE subtrees.  Please take care of them before
calling
void PathExpNode::becomeOperatorNode( NodeTypeID newType,
PathExpNode*i_lhs,
                                      PathExpNode*i_rhs){
   assert(newType != PTH);
   NodeTypeID originalType = typeID();

   if(originalType == PTH){
      isNegative(0);
      edgeList().erase(edgeList().begin(), edgeList().end()); }

   typeID(newType); lhs(i_lhs); rhs(i_rhs);

}


////////////////////////////////////////////////////////////////////////
///
//Print Methods -- these methods take the place of virtual print
function//
////////////////////////////////////////////////////////////////////////
///

void PathExpNode::PTH_printOn(ostream& out){
   assert(typeID()==PTH);

        if(isNegative()) out << "(-";

   unsigned long numEdges=edgeList().size();
   EdgeList::const_iterator iter=edgeList().begin();
        unsigned long i=1;

   for(i=1; i < numEdges; ++i, ++iter)
      out << *iter << "*";
   if(i==numEdges)
      //Print last edge without * (note that i starts at 1 not 0)
      out << *iter;

        if(isNegative()) out << ")";
}

void PathExpNode::CP_printOn(ostream& out){
   assert(typeID()==CP);
   out << "(";
   lhs()->printOn(out); out << " (+) "; rhs()->printOn(out);
   out<<")";}
```

```
void PathExpNode::CT_printOn(ostream& out){
   assert(typeID()==CT);
   out << "(";
   lhs()->printOn(out); out << " (*) "; rhs()->printOn(out);
   out<<")";}

void PathExpNode::PL_printOn(ostream& out){
   assert(typeID()==PL);
   out << "(";
   lhs()->printOn(out); out << " + "; rhs()->printOn(out);
   out<<")";}

void PathExpNode::MI_printOn(ostream& out){
   assert(typeID()==MI);
   out << "(";
   lhs()->printOn(out); out << " - "; rhs()->printOn(out);
   out<<")";}


//Prints this expression without most of the parentheses.
//As yet, does not understand anything about operator precedence
//or associativity.  Use at your own risk (when parentheses are
//known not to be needed)
void PathExpNode::printOnWithoutParentheses(ostream& out){
   switch (typeID()){
      case PTH: PTH_printOn(out); break;

      case CP: CP_printOnWithoutParentheses(out); break;

      case CT: CT_printOnWithoutParentheses(out); break;

      case PL: PL_printOnWithoutParentheses(out); break;

      case MI: MI_printOnWithoutParentheses(out); break;

      default:
         cerr <<"Unknown typeID:"<<typeID()<<endl; assert(0);
   };
}


void PathExpNode::CP_printOnWithoutParentheses(ostream& out){
   assert(typeID()==CP);
   lhs()->printOnWithoutParentheses(out);
        out << " (+) ";
        rhs()->printOnWithoutParentheses(out); }

void PathExpNode::CT_printOnWithoutParentheses(ostream& out){
   assert(typeID()==CT);
   lhs()->printOnWithoutParentheses(out);
        out << " (*) ";
        rhs()->printOnWithoutParentheses(out); }
```

```
void PathExpNode::PL_printOnWithoutParentheses(ostream& out){
   assert(typeID()==PL);
   lhs()->printOnWithoutParentheses(out);
        out << " + ";
        rhs()->printOnWithoutParentheses(out); }

void PathExpNode::MI_printOnWithoutParentheses(ostream& out){
   assert(typeID()==MI);
   lhs()->printOnWithoutParentheses(out);
        out << " - ";
        rhs()->printOnWithoutParentheses(out); }
```

```
#include "PathExpression.h"

PathExp::PathExp(BarePathList const & list){
        BarePathList::const_iterator iter=list.begin();

        if(list.size()>0){
                EdgeList l(*iter);
                l.sort();
                m_root=new PathExpNode(PTH, false, l);
                ++iter;
        } else {
                EdgeList empty; m_root=new PathExpNode(PTH, false,
empty); }

        for(unsigned long i=1; i<list.size(); ++i, ++iter){
                EdgeList l(*iter);
                l.sort();
                root()->appendEdgeList(CP, l);
        }
}

//Given that this node is a CirclePlus, removes CirclePlusses from it
//and all its subtrees
void PathExpNode::CP_removeCirclePlus(){
        assert(typeID() == CP);

        lhs()->removeCirclePlus();

        PathExpNode *oldLHS=lhs(), *oldRHS=rhs();

        PathExpNode *newLHS= new PathExpNode(PL, oldLHS, oldRHS);
        PathExpNode *newRHS=
                new PathExpNode(CT, new PathExpNode(*oldLHS), new
PathExpNode(*oldRHS));

        becomeOperatorNode(MI, newLHS, newRHS);

        rhs()->removeCirclePlus();
}

//Given that this node is a CircleTimes, removes CircleTimeses from it
//and all its subtrees
void PathExpNode::CT_removeCircleTimes(){
        assert(typeID() == CT);

        //note: didn't test any situations in which this could be true.
        //however, this takes care of the case of an operator over which
        //circleTimes can't distribute -- since circleTimes is the only
such
        //operator.
        if(lhs()->typeID() == CT) lhs()->removeCircleTimes();
        if(rhs()->typeID() == CT) rhs()->removeCircleTimes();
```

```
        bool lhsPath = lhs()->typeID() == PTH;
        bool rhsPath = rhs()->typeID() == PTH;

        if(lhsPath)
                if(rhsPath){
                        CT_removeCircleTimesPathPath();
                        return;
                }else
                        CT_removeCircleTimesPathOper();
        else
                if(rhsPath){
                        //Swap two sides
                        PathExpNode* tmp=lhs();
                        lhs(rhs()); rhs(tmp);

                        CT_removeCircleTimesPathOper();
                }else
                        CT_removeCircleTimesOperOper();

        lhs()->removeCircleTimes();
        rhs()->removeCircleTimes();
}

template<class T>
bool is_sorted(list<T> const&l){
        list<T>::const_iterator iter=l.begin(), lastIter=iter;
        ++iter;
        for(unsigned long i=1; i<l.size(); ++i, ++iter)
                if(*iter<*lastIter) return(false);
        return(true);
}

//Given that this node is a CircleTimes
//And given that both children of this node are paths,
//Changes this node into a path whose edges are the union of the edges
in
//the children.
void PathExpNode::CT_removeCircleTimesPathPath(){

        assert(typeID() == CT);
        EdgeList const &lhsEdges=lhs()->edgeList();
        EdgeList const &rhsEdges=rhs()->edgeList();

        assert(is_sorted(lhsEdges));
        assert(is_sorted(rhsEdges));

        EdgeList newEdges;

        //Doesn't seem to work under old gnu compiler
        //set_union(lhsEdges.begin(), lhsEdges.end(),
        //          rhsEdges.begin(), rhsEdges.end(), newEdges.begin());
```

```
        newEdges=lhsEdges;
        EdgeList::const_iterator item=rhsEdges.begin();
        while(item != rhsEdges.end())
                newEdges.push_back(*item++);
        newEdges.sort();

        //Doing a unique
        EdgeList::iterator cur=newEdges.begin();
        EdgeList::iterator prev=cur;
        ++cur;
        while(cur != newEdges.end()){
                assert(prev != cur);
                if(*prev!=*cur){
                        prev=cur;
                        ++cur;
                }else{
                        EdgeList::iterator tmp=cur;
                        ++cur;
                        newEdges.erase(tmp);
                }
        }


        bool isNeg=lhs()->isNegative();
        if(rhs()->isNegative() ) isNeg = !isNeg;

        delete lhs(); delete rhs();
        becomePathNode(isNeg, newEdges);
}

//Given that this node is a CircleTimes
//And given that the left hand side is a path and the right hand side
//is an operator over which CircleTimes distributes, distributes the
//circleTimes over oper.
void PathExpNode::CT_removeCircleTimesPathOper(){
        assert(typeID() == CT);

        PathExpNode *path=lhs(),
                    *oldRHS=rhs();
        NodeTypeID   oldRHSOperator=rhs()->typeID();

        PathExpNode *newLHS=new PathExpNode(CT, path, oldRHS->lhs());

        oldRHS->becomeOperatorNode(CT, new PathExpNode(*path), oldRHS-
>rhs());

        becomeOperatorNode( oldRHSOperator, newLHS, oldRHS );
}

//Given that this node is a CircleTimes
//And given that both sides are operators and that the left hand side
//is an operator over which CircleTimes distributes, distributes the
```

```
//CircleTimes over oper.
void PathExpNode::CT_removeCircleTimesOperOper(){
        assert(typeID() == CT);

        //Simplifying expression of form
        //  (A ? B) (*) (C ?? D)
        //into
        //  (A (*) (C ?? D)) ? (B (*) (C ?? D))
        // This has the effect of moving the (*) down a level so it will
eventually
        // reach the bottom of the tree -- composed of paths.

        PathExpNode *a= lhs()->lhs(),
                    *b= lhs()->rhs(),
                    *cd= rhs(), //The expression (C ?? D)
                    *cdCopy= new PathExpNode (*cd);

        becomeOperatorNode(lhs()->typeID(), lhs(), rhs());

        lhs()->becomeOperatorNode(CT, a, cd);
        rhs()->becomeOperatorNode(CT, b, cdCopy);
}

//Removes all minuses in this tree and its subtrees
void PathExpNode::removeMinus(){
        switch(typeID()){
                case MI:
                        becomeOperatorNode(PL, lhs(), rhs());
                        lhs()->removeMinus();
                        rhs()->removeMinusAndNegate();
                break;

                case PTH: break;

                case CT: case CP: case PL:
                        lhs()->removeMinus();
                        rhs()->removeMinus();
                break;

                default:
                        cerr <<"Unknown typeID:"<<typeID()<<endl;
assert(0);
        }
        return;
}

//Removes all minuses in this tree and its subtrees while negating every
//path which it encounters.
void PathExpNode::removeMinusAndNegate(){
        switch(typeID()){
                case MI:
                        becomeOperatorNode(PL, lhs(), rhs());
```

```
                        lhs()->removeMinusAndNegate();
                        rhs()->removeMinus();
                break;

                case PTH: isNegative(!isNegative()); break;

                case CT: case CP: case PL:
                        lhs()->removeMinusAndNegate();
                        rhs()->removeMinusAndNegate();
                break;

                default:
                        cerr <<"Unknown typeID:"<<typeID()<<endl;
assert(0);
        }
        return;
}

//Appends the path specified by edgeList to the expression at this
point,
//joined by an operator of type operatorType.
//BAD THINGS if operatorType==PTH
void PathExpNode::appendEdgeList(NodeTypeID operatorType,
                                 EdgeList const &i_edgeList){
   assert(operatorType != PTH);

   PathExpNode *newNode=NULL;
   if(typeID() == PTH){
      newNode=new PathExpNode(typeID(), isNegative(), edgeList());
   }else{
      newNode=new PathExpNode(typeID(), lhs(), rhs());
   }

   PathExpNode *newPath=new PathExpNode(PTH, false, i_edgeList);

   becomeOperatorNode(operatorType, newNode, newPath);
}

////////////////////////////////////////////////////////////////////
//Become Methods -- allow the object to change its type (magic :)//
////////////////////////////////////////////////////////////////////
//Become path function
//Note: DOES NOT DELETE subtrees.  Please take care of them before
calling
//Note: does not always use the accessors since they are picky about
values.
void PathExpNode::becomePathNode(bool i_isNegative, EdgeList const&
i_edgeList){
   NodeTypeID originalType = typeID();

   if(originalType != PTH){ m_lhs=NULL; m_rhs=NULL; };
```

```
   typeID(PTH); isNegative(i_isNegative); edgeList()=i_edgeList;
}

//Become operator function -- BAD THINGS happen if newType == PTH
//Note: DOES NOT DELETE subtrees.  Please take care of them before
calling
void PathExpNode::becomeOperatorNode( NodeTypeID newType,
PathExpNode*i_lhs,
                                      PathExpNode*i_rhs){
   assert(newType != PTH);
   NodeTypeID originalType = typeID();

   if(originalType == PTH){
      isNegative(0);
      edgeList().erase(edgeList().begin(), edgeList().end()); }

   typeID(newType); lhs(i_lhs); rhs(i_rhs);

}


////////////////////////////////////////////////////////////////////////
///
//Print Methods -- these methods take the place of virtual print
function//
////////////////////////////////////////////////////////////////////////
///

void PathExpNode::PTH_printOn(ostream& out){
   assert(typeID()==PTH);

        if(isNegative()) out << "(-";

   unsigned long numEdges=edgeList().size();
   EdgeList::const_iterator iter=edgeList().begin();
        unsigned long i=1;

   for(i=1; i < numEdges; ++i, ++iter)
      out << *iter << "*";
   if(i==numEdges)
      //Print last edge without * (note that i starts at 1 not 0)
      out << *iter;

        if(isNegative()) out << ")";
}

void PathExpNode::CP_printOn(ostream& out){
   assert(typeID()==CP);
   out << "(";
   lhs()->printOn(out); out << " (+) "; rhs()->printOn(out);
   out<<")";}
```

```
void PathExpNode::CT_printOn(ostream& out){
   assert(typeID()==CT);
   out << "(";
   lhs()->printOn(out); out << " (*) "; rhs()->printOn(out);
   out<<")";}

void PathExpNode::PL_printOn(ostream& out){
   assert(typeID()==PL);
   out << "(";
   lhs()->printOn(out); out << " + "; rhs()->printOn(out);
   out<<")";}

void PathExpNode::MI_printOn(ostream& out){
   assert(typeID()==MI);
   out << "(";
   lhs()->printOn(out); out << " - "; rhs()->printOn(out);
   out<<")";}


//Prints this expression without most of the parentheses.
//As yet, does not understand anything about operator precedence
//or associativity.  Use at your own risk (when parentheses are
//known not to be needed)
void PathExpNode::printOnWithoutParentheses(ostream& out){
   switch (typeID()){
      case PTH: PTH_printOn(out); break;

      case CP: CP_printOnWithoutParentheses(out); break;

      case CT: CT_printOnWithoutParentheses(out); break;

      case PL: PL_printOnWithoutParentheses(out); break;

      case MI: MI_printOnWithoutParentheses(out); break;

      default:
         cerr <<"Unknown typeID:"<<typeID()<<endl; assert(0);
   };
}


void PathExpNode::CP_printOnWithoutParentheses(ostream& out){
   assert(typeID()==CP);
   lhs()->printOnWithoutParentheses(out);
        out << " (+) ";
        rhs()->printOnWithoutParentheses(out); }

void PathExpNode::CT_printOnWithoutParentheses(ostream& out){
   assert(typeID()==CT);
   lhs()->printOnWithoutParentheses(out);
        out << " (*) ";
        rhs()->printOnWithoutParentheses(out); }
```

```
void PathExpNode::PL_printOnWithoutParentheses(ostream& out){
   assert(typeID()==PL);
   lhs()->printOnWithoutParentheses(out);
        out << " + ";
        rhs()->printOnWithoutParentheses(out); }

void PathExpNode::MI_printOnWithoutParentheses(ostream& out){
   assert(typeID()==MI);
   lhs()->printOnWithoutParentheses(out);
        out << " - ";
        rhs()->printOnWithoutParentheses(out); }
```

Bibliography

1. Alexopoulos, Christos, and Fishman, George S. "Characterizing Stochastic Flow Networks Using the Monte Carlo Method," *Networks*, 21: 775-798 (1991).
2. Alexopoulos, Christos, and Fishman, George S. "Sensitivity Analysis in Stochastic Flow Networks Using the Monte Carlo Method," *Networks*, 23: 605-621 (1993).
3. *The American Heritage College Dictionary* (Third Edition). New York: Houghton Mifflin Company, 1993.
4. Bagga, K. S., Beineke, L. W., Pippert, R. E., and Lipman, M. J. "A Classification Scheme for Vulnerability and Reliability Parameters of Graphs," *Mathl. Comput. Modelling*, 17(11): 13-16 (1993).
5. Bazaraa, M. O., Jarvis, J. J., and Sherali, H. D. *Linear Programming and Network Flows*. New York: John Wiley and Sons, 1990.
6. Cave, Jonathan. "Introduction to Game Theory," RAND Paper P-7336, April 1987.
7. Chan, Yupo. Class Notes, OPER 621, Multiple Criteria Decision Making, School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB OH, Spring 1997.
8. Chan, Yupo. *Facility Location and Land Use—Multi-criteria Analysis of Spatial-Temporal Information*. Draft Book, 1997.

9. Chan, Y., Jansen, L. J., Gaught, W., Borsi, J., and Kissin, S. "Multicriteria Stochastic-Network Optimization: Improving Reliability vs. Throughput," Department of Operational Sciences, Air Force Institute of Technology, Working Paper Series, WP96-08, October 1996.

10. Chan, Yupo, Yim, Eugene, and Marsh, Alfred. "Exact and Approximate Improvement to the Throughput of a Stochastic Network," Draft Paper, Air Force Institute of Technology, April 1997.

11. Colbourn, Charles J. *The Combinatorics of Network Reliability*. New York: Oxford University Press, 1987.

12. Gerakoulis, D. P. "A Parameter for Vulnerability Evaluation of Packet Switched Communications Networks," IEEE Global Telecommunications Conference, GLOBECOM '91 (1991).

13. de Sa, Derek. *Vulnerability of Flow Networks*. Ph.D. Dissertation, Department of Management Science and Statistics, University of Alabama, 1989.

14. "Movers and Shakers," *The Economist*, 24 January, 1998: 63.

15. Goddard, Wayne. "Measures of Vulnerability—The Integrity Family," *Networks*, 24: 207-213 (1994).

16. Harms, Daryl D., Kraetzl, Miro, Colbourn, Charles J., Devitt, John S. *Network Reliability: Experiments with a Symbolic Algebra Environment*. New York: CRC Press, 1994.

17. Jansen, Leonard J. *Improving Stochastic Communication Network Performance: Reliability vs. Throughput.* MS Thesis, AFIT/GSO/ENS/91D-10. School of Engineering, Air Force Institute of Technology (AU), December 1991 (AD-A243655).

18. Lai, Hong-Jian. "Large Survivable Nets and the Generalized Prisms," *Discrete Applied Mathematics*, 61: 181-185 (1995).

19. Liew, Soung C. "A Framework for Characterizing Disaster-Based Network Survivability," *IEEE Journal on Selected Areas in Communications*, 12(1): 52-57 (1994).

20. *LINGO User's Guide.* Chicago: Lindo Systems Inc., 1995.

21. Lyle, David L. *Analyzing and Improving Stochastic Network Security: A Multicriteria Prescriptive Risk Analysis Model.* MS Thesis, AFIT/GOR/ENS/97M-15. School of Engineering, Air Force Institute of Technology (AU), March 1997.

22. Malashenko, Y. Y., Rogozhin, V. S., and Ferapontov, Y. V. "Deterministic Models for Estimating Vulnerability of Networks," *Soviet Journal of Computer and Systems Sciences*, 27(4): 125-135 (1989).

23. Morris, Peter. *Introduction to Game Theory.* New York: Springer-Verlag, 1991.

24. Schwartau, Winn. *Information Warfare.* New York: Thunder's Mouth Press, 1996.

25. Sengoku, Masakazu, Shinoda, Shoji, and Yatsuboshi, Reigo. "On a Function for the Vulnerability of a Directed Flow Network," *Networks*, 18: 73-83 (1988).

26. Shi, Jianxu, and Fonseka, John P. "Traffic-Based Survivability Analysis of Telecommunications Networks," IEEE/IEICE Global Telecommunications Conference, 1995.

27. Shier, Douglas R. *Network Reliability and Algebraic Structures*. Oxford: Clarendon Press, 1991.

28. Steuer, Ralph, E. *Multiple Criteria Optimization: Theory, Computation, and Application*. New York: John Wiley & Sons, 1986.

29. Winston, W. L. *Operations Research: Applications and Algorithms*. Belmont, California: International Thomson Publishing, 1994.

30. Yim, Eugene. *Improving the Survivability of a Stochastic Communication Network*. MS Thesis, AFIT/GOR/ENS/88D-25. School of Engineering, Air Force Institute of Technology (AU), December 1988 (AD-A202872).

31. Zolfaghari, Ali, and Kaudel, Fred J. "Framework for Network Survivability Performance," *IEEE Journal on Selected Areas in Communications*, 12(1): 46-51 (1994).

Vita

Captain Jeffrey A. Schavland was born to Endre E. and Vernadel M. Schavland on 9 June 1969 in Rockford, Illinois, USA. He earned a Bachelor of Science degree in Aeronautical and Astronautical Engineering from the University of Illinois at Urbana-Champaign in 1992. During 1990-91, Captain Schavland attended the *Technische Universität München* as a German Academic Exchange Service Scholar. He received his commission on 18 May 1992.

Before arriving at AFIT in August 1996, Captain Schavland was a developmental engineer at the Countermeasures Hands-On Program (CHOP), Space and Missiles Technology Division, Phillips Laboratory, Kirtland AFB, New Mexico.

As a follow-on to his study at AFIT, Captain Schavland has been assigned to the Defense Logistics Agency, where he will be an analyst at the DLA Office of Operations Research and Resource Analysis (DORRA) at the Defense Supply Center Richmond (Virginia).

Captain Schavland is married to the former Adrienn Pehl of Budapest, Hungary.

Permanent Address:	1395 Country Farm Road, Rockford IL 61107, USA.
Electronic mail:	Schavland@aol.com

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1998	Master's Thesis

4. TITLE AND SUBTITLE
A Game-Theoretic Model for Stochastic Network Improvement: Reliability vs. Throughput

5. FUNDING NUMBERS

6. AUTHOR(S)
Jeffrey A. Schavland, Capt, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology
2950 P Street
Wright-Patterson AFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GOR/ENS/98M-22

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Academic Advisor: Professor Yupo Chan
Telephone: 937/255-6565 x4331
E-mail: ychan@afit.af.mil

12a. DISTRIBUTION AVAILABILITY STATEMENT
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Prescriptive models used to allocate resources for network improvement traditionally have used reliability or flow as measures of effectiveness (MOEs). Such metrics do not give value to efforts which make a component more difficult to exploit. This study developed an entirely new MOE for stochastic network improvement, flow damage utility, which uses a two-person, zero-sum, non-cooperative game to optimize a probabilistic network for an estimate of expected flow minus performance degradation after a worst case component loss. A multiple criteria optimization problem that uses flow damage utility and an analogous, previously developed metric for the reliability problem is used to capture the strategic competition between the network defender and attacker and shows promise of finding "value free" improvement/defensive strategies in the context of Steuer's reverse filtering as applied to the generated efficient frontier. Irrespective of unit costs of reliability vs. bandwidth improvement, a "value free" solution may be imputed from these game-theoretic models. Examples of analysis on four different networks are presented.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Networks, Stochastic Networks, Communication Networks, Game Theory, Reliability, Vulnerability, Survivability, Multiple Criteria Optimization, Maximum Flow	189
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Standard Form 298 (Rev. 2-89) (EG)
Prescribed by ANSI Std. 239.18
Designed using Perform Pro, WHS/DIOR, Oct 94